NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

H82975

AFFINE INVARIANT OBJECT RECOGNITION BY
VOTING MATCH TECHNIQUES

by

Hsu, Tao-I

December 1988

Thesis Advisor: Chin-Hwa Lee

Approved for public release; distribution is unlimited

T241973

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) Naval Postgraduate School	5 MONITORING ORGANIZATION REPORT NUMBER(S) Naval Postgraduate School
6a NAME OF PERFORMING ORGANIZATION Monterey, California 6b OFFICE SYMBOL (If applicable) 62	7a NAME OF MONITORING ORGANIZATION Monterey, California
6c ADDRESS (City, State, and ZIP Code)	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
AFFINE INVARIANT OBJECT RECOGNITION BY VOTING MATCH TECHNIQUES

12 PERSONAL AUTHOR(S)
HSU, Tao-I

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1988 December	103

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Affine transformation; hashing function

19 ABSTRACT (Continue on reverse if necessary and identify by block number)
This thesis begins with a general survey of different model based systems for object recognition. The advantage and disadvantage of those systems are discussed. A system is then selected for study because of its effective Affine invariant matching [Ref. 1] characteristic. This system involves two separate phases, the modeling and the recognition. One is done off-line and the other is done on-line. A Hashing technique is implemented to achieve fast accessing and voting. Different test data sets are used in experiments to illustrate the recognition capabilities of this system. This demonstrates the capabilities of partial match, recognizing objects under similarity transformation of applied to the models and the results of noise perturbation. The testing results are discussed, and related experiences and recommendations are presented.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL Chin-Hwa Lee	22b TELEPHONE (Include Area Code) 408-646-2190	22c OFFICE SYMBOL 62Le

DD Form 1473, JUN 86 *Previous editions are obsolete* SECURITY CLASSIFICATION OF THIS PAGE

Approved for public release; distribution is unlimited.

Affine Invariant Object Recognition by Voting Match Techniques

by

Hsu Tao-i
Captain, Taiwan Republic of China Army
B.S. Math, Chung Ching Institute of Technology, 1982

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1988

ABSTRACT

This thesis begins with a general survey of different model based systems for object recognition. The advantage and disadvantage of those systems are discussed. A system is then selected for study because of its effective Affine invariant matching [Ref. 1] characteristic. This system involves two separate phases, the modeling and the recognition. One is done off-line and the other is done on-line. A Hashing technique is implemented to achieve fast accessing and voting. Different test data sets are used in experiments to illustrate the recognition capabilities of this system. This demonstrates the capabilities of partial match, recognizing objects under similarity transformation applied to the models, and the results of noise perturbation. The testing results are discussed, and related experiences and recommendations are presented.

C.1

TABLE OF CONTENTS

I. INTRODUCTION 1

A. FEATURE EXTRACTION (PREPROCESSING) 2

1. Fourier Descriptor Technique 3

2. Moment Techniques 3

3. Technique of Accumulating Local Evidence by Clustering 4

B. MODELING (TRAINING/LEARNING) 4

1. Radius-angle 5

2. Orientation-arc Length 5

3. Curvature-arc Length 6

C. MATCHING (RECOGNITION) 6

1. Statistical Matching 7

2. Syntactic/Structural Technique 9

3. Relational Graph Method 10

4. Voting Match Techniques 11

II. AFFINE INVARIANT MATCHING 15

A. PARTIAL MATCHING 15

B. AFFINE TRANSFORMATION AND VOTING TECHNIQUE 16

1. Affine Coefficient 16

2. Affine Coefficient Transform 18

C. AFFINE INVARIANT RECOGNITION ALGORITHM . . . 22
1. Preprocessing (Feature Extraction) . . . 23
2. Data-Base Setups (Modeling) 24
3. Voting Match 25
III. ALGORITHM IMPLEMENTATION 28
A. SEARCHING 28
B. HASH TECHNIQUE 28
1. Hashing Implementation 30
C. PREPROCESSING 31
1. Laplacian Operator EGLP [Ref. 10] 31
2. Threshold Operation SLTH 31
3. Boundary Thinning THNG [Ref. 10] 32
4. Tracing Boundary BDFL [Ref. 10] 32
5. Maximum Curvature Finding MAXCUR 33
D. DATA-BASE SETUP 33
1. Affine Transformation 33
2. Hash Table Generation 37
E. RECOGNITION 40
1. SEARCH Module 43
2. VERIFY Module 44
3. RECONSTRUCT Module 44
IV. RESULTS AND PERFORMANCE 46

A. EXPERIMENT RESULTS 46

1. Test For Similarity Transformation . . . 46

2. Test For Partial Matching 48

3. When Numerical Error is Present . . . . . in the Test Object 49

B. ALGORITHM PERFORMANCE AND COMPLEXITY 50

C. LIMITATIONS 53

1. Model Representation 53

2. Quantization Problem in Generation . . . . of the Hash Key 54

3. Noise Handling 54

D. IMPRESSIONS OF THE ALGORITHM FEATURE 55

1. Two Phase Algorithm 55

2. Hash Implementation 55

V. CONCLUSION 57

A. SUMMARY 57

B. EXPERIENCE GAINED 58

1. Affine Invariant 58

2. Partial Matching 58

3. Speed and Complexity 58

4. Noise Perturbation 59

C. RECOMMENDATIONS 59

APPENDIX A: PASCAL-LIKE PSEUDO CODES 61

APPENDIX B: LISTING OF SOURCE CODES. 68

LIST OF REFERENCES 93

INITIAL DISTRIBUTION LIST 94

ACKNOWLEDGMENTS

I would like to express my deep appreciation to my Thesis Advisor, Dr. Chin-Hwa Lee, for his guidance and counsel in assisting in the completion of this thesis.

I would also like to thank Dr. C.W. Therrien and LT Michael Shields and Mrs. Cindy Shields, and Dan Zulaica who contributed their assistance in the accomplishment of this thesis.

Finally, I wish to express my gratitude to my parents, and my fiancee, Wen-Yao Ko, who helped and supported me through these laborious times to achieve this educational goal.

I. INTRODUCTION

There has been considerable growth of interest in problems of object recognition. This interest has created an increased amount of theoretical methods and experimental software/hardware for the design of object recognition systems. Applications of object recognition include character recognition, target detection, medical diagnosis, analysis of biomedical images, remote sensing, identification of human faces or fingerprints, archaeology, speech recognition, machine part recognition, and automatic inspection.

Object recognition is primarily concerned with the description and classification of measurements taken from some physical or mental process. In general, object recognition may be considered as:

- Observing the attributes or characteristics of the objects,
- Selecting useful features from the set of characteristics that are expected for representation of the objects,
- Performing the matching procedure with respect to a specific goal on the basis of the representation.

It is obvious that the most popular and largely used technique in object recognition is model-based recognition. A model-based recognition system involves comparing the input image with a set of predefined models of objects. The goal

of such a system is to create a description of each of a known set of objects in advance. Then, these object models are used to recognize unknown objects in an image and to specify position and orientation relative to the viewers. A model-based vision system contains three basic phases: *feature extraction, object modeling, and unknown object matching.* These are discussed in the three subsections that follow.

A. FEATURE EXTRACTION (PREPROCESSING)

Using effective features in object recognition is a very important factor for success. Features can be used to describe or synthesize objects. Features of an object are most often boundaries and geometric measurements derived from boundaries. These features can be roughly classified into three types: global, local, and relational features.

- Global features include area (size), perimeter, centroid, distance of contour points from the centroid, curvature and moment of interia, and others which provide useful information for object manipulation. The representation of an object with these global features can be stored in a feature lists, and the *ordering* of the features in the feature list is not important.

- Local features include line segments, arc segments with constant curvature and corner, each describing a portion of the object. They are organized in a highly structured manner, such as an ordered list or a sequence of equations. The *ordering* of the feature in this highly structured manner is usually related to the object's boundary.

- Relational features include a variety of distance and relative orientation measurements relating substructures and regions of an object. Geometric relations among

local features (e.g., corner and line) may be of particular interest.

Typical feature extraction techniques are the following:

1. Fourier Descriptor Technique

Fourier series representation of the parameterized boundary is one of the oldest and most well known transform technique. A finite number of harmonics of the Fourier descriptors are computed from the pattern boundary and compared with a set of reference Fourier descriptors. This description is global in nature, i.e., each coefficient depends on every other point of the boundary. It is therefore not suitable for recognition of partially occluded objects. A minimum-distance classification rule can be used for the recognition of various parts. [Ref. 2]

2. Moment Techniques

Moment techniques use parameters calculated from the pattern interior points. In these techniques, coordinates of points belonging to the pattern are used to compute a set of moments. These moments can be normalized to obtain measures that are invariant under scaling and rotation [Ref. 3]. It is difficult to relate higher order moments to the pattern. This requires global feature extraction, so the *moment techniques* have a shortcoming in recognizing partially occluded objects.

3. Technique of Accumulating Local Evidence by Clustering

Directed edge elements (vectors) are used as one type of primary feature which contains directional, positional, and size information. First, point features (i.e., the head and tail of a vector) are extracted, and then vectors are formed from suitable point pairs. Straight edge detectors, curved edge detectors, circle detectors, and intersection detectors are employed to define vectors between point pairs. Holes are detected by a set of circular masks and curves and intersections are detected by linking edges together [Ref. 4].

Overall, using local features has the following advantages:

- Local features may be cheaper to compute because they are simpler and can be selectively (sequentially) detected;
- If a few local features are missing (due to noise or occlusion), it may still be possible to recognize the object on the basis of the remaining features associated with the object;
- Since a few types of local features are often sufficient to describe a large number of complex objects, it is possible to specify only a few number of local feature detectors which can be applied to the image.

B. MODELING (TRAINING/LEARNING)

Modeling is based on the object's geometric properties such as object's shape and surface relative to the viewing angle. The three basic types of features (the global feature, the local feature, and the relational feature) are then employed to describe or represent objects.

Possible modeling schemes include *radius-angle* representation, *orientation-arc length* representation, *curvature-arc length* representation.

1. Radius-angle

The radius-angle representation requires a reference origin. This is usually taken to be the object's centroid. This representation is obviously scale-dependent. The need for a reference origin (global feature) also makes it unsuitable for recognizing partially occluded objects. Also the need for the representation to be single-valued further restricts the type of shapes that can be modeled in this manner.

2. Orientation-arc Length

The orientation-arc length representation models the angle made between a fixed axis and a tangent to the boundary as a function of the arc length. This representation is scale invariant but not orientation invariant. Straight horizontal lines or this representation correspond to zero curvature (i.e., straight lines in the boundary), and straight non-horizontal lines correspond to segments of circles with the radii or curvature given by the slopes of the lines. (This allows the boundary to be easily segmented into straight lines and circular arcs and is used sometimes in the initial processing for feature matching).

3. Curvature-arc Length

The curvature-arc length representation models the curvature of the boundary as a function of arc length. This representation is orientation-invariant but unfortunately it is not scale independent. (A circle of radius r, for example, has a curvature of 1/r). Also, curvature is very sensitive to noise. However, curvature is a popular descriptor and this representation is often used to extract the extremas (in curvature) for feature matching [Ref. 5].

A discrete version of orientation-arc length representation has also been used. Commonly called chain codes, this models the boundary in short line segments that lie on a fixed grid with a fixed set of orientations. Although efficient in representation and cross-matching, chain codes are rather sensitive to noise and have other shortcomings that make this representation unsuitable for general shape matching.

It should be noted that none of the representations discussed above is simultaneously scale and orientation-invariant.

C. MATCHING (RECOGNITION)

Given a set of models that describe all aspects of an object to be recognized, the process of model-based recognition consists of matching features extracted from

unknown objects of a given input image with respect to those of the models. Matching techniques using global, local or relational features provide a way to recognize and locate a part on the basis of a few key features. Therefore, the model description dominates other procedures in the model-based matching process. The choice of matching process is highly dependent on the type of model used for object representation.

We divide matching processes into three schemes: *statistical matching*, *graph matching*, and *syntactic (structural) matching*. Models using global features are usually associated with statistical matching techniques. Models based on local features are usually associated with syntactic matching techniques, and models using both local and relational features are usually associated with graph matching techniques. Table 1 summarizes the techniques used in three phases of the model-based matching procedure.

1. Statistical Matching

Statistical pattern recognition can be divided into non-parametric and parametric methods. Non-parametric classification uses separation of clusters in feature space and to recognize pattern classes. Parametric classification on the other hand, is based on Bayes rule. It states that the input feature vector belongs to a particular class, say j, if the likelihood ratio (λ) between two pattern classes i and j

Table 1. THE THREE METHODS BASED ON 2-D IMAGE REPRESENTATIONS.

PHASE / TECHNIQUES	Feature extraction	Modeling	Matching
	Global scalar	Feature vector (unordered)	Statistical pattern recognition
	Local	Ordered string of features or abstract description of feature string	Syntactical or voting verification of string description
	Local and relational	Relational graph	Graph searching

is greater than the ratio of the probabilities of occurrence of the pattern classes j over i.

For example, consider two classes ω_1 and ω_2 with *a priori* probabilities P_1 and P_2 where $(P_1 + P_2 = 1)$. The Bayes discriminant function $f_{12}(\chi)$ defining the decision boundary between class 1 and 2 has the form

$$f_{12}(x) = \frac{1}{p(x)}[P_1 L(\omega_1, d_2) p(x/\omega_1) - P_2 L(\omega_2, d_1) p(x/\omega_2)] \geq 0 \qquad (1)$$

where

$$L(\omega_i, d_j) i,j = 1,2, \qquad (2)$$

is the loss incurred with the decision d_j when the i-th class is true, and

$$p(x) = \sum_{i=1}^{2} P_i p(x/\omega_i) \qquad (3)$$

and $p(\chi/\omega_1)$ and $p(\chi/\omega_2)$ represent the conditional probability densities associated with measurement χ, given the unknown pattern that came from class ω_1 or ω_2 respectively. If $f_{12}(\chi) \geq 0$, we decide that χ belongs to the 1st class; if $f_{12}(\chi) < 0$, decide the 2nd class.

It is evident that Bayes rule is based on *a priori* knowledge. In practice, the collection of all the *priori* statistical data becomes a serious problem.

2. Syntactic/Structural Technique

In syntactic methods, an object model is represented by using abstracted and precise geometric primitives such as arcs, lines and corners. These primitives are local in nature, each describing a portion of the object. They are organized in a highly structured manner, such as an ordered list or a sequence of equations. The ordering of primitives in this type of method is usually related to the object's boundary in such a way that following the entire primitive list sequentially is equivalent to tracing the boundary of the object. Recognition uses a hypothesis-verification procedure. The structural local primitives of the model are used to predict where objects are located in the scene. Then,

primitives of the hypothesized object are measures on the basis of the prediction hypothesized in the model, in order to verify and find the match.

In this type of matching, the local primitives are transformed into primitives which are organized into strings (sentences) by some highly structured grammatical rules. Matching is performed by parsing. A major problem with the grammatical model is the construction of a grammar that is comprehensive enough to generate all the possible types of shapes of interest and yet discriminatory enough to reject others. The grammatical model is inherently one-dimensional. Noise perturbation of the model is also detrimental. A number of grammars have been developed over the years. A good description of these can be found in Reference 6.

3. Relational Graph Method

Objects can be represented structurally by graphs. In this method, relationships among local geometric primitives are represented by a graph. In the graph each node represents a local geometric primitive and is labeled with a list of properties (e.g., size) of the geometric primitive. Arcs represent relational primitive linking pairs of nodes and are labeled with lists of relation values (e.g., distance and adjacency). Recognition of the object becomes a graph-matching process. A disadvantage with this type of method is the fact that a large number of geometric primitives must be

detected and grouped together to recognize an object. Thus the matching algorithm used with these models must be more complex and may be slower than matching algorithms used with the other methods. Noises in the geometric primitives may change the graph of the unknown object. No precise analytic treatment of the noise problem is currently known.

4. Voting Match Techniques

A technique which is different from the traditional techniques discussed above is the *Voting Match* techniques. In this technique an object's model is defined first by transforming the object's local features into an abstract vector set. That is, the model representation is condensed to lower dimensional vector sets which preserve the geometrical characteristics corresponding to the original feature vectors. Matching is done using an accumulator cell to collect the strength of instances of occurring objects, that is, to increment a vote whenever a match between the unknown scene and the model occurs. The procedure matches all possible instances of the image and the model features on the basis of local evidence. From the peak strength in the accumulator cell the candidacy of the model can be picked up. Then, for verification, a line-drawing version of the object will be performed. This method is believed to be more robust because the voting procedure integrates all local information before any recognition decision is made.

There are two typical voting match techniques. These are Hough transform method and Affine transform method. Both use geometric transformations to map instances of a given pattern into peaks of a transform space.

The Hough transform method was originally developed to handle simple pattern such as straight lines and circles, but it was recently extended to arbitrary shapes [Ref. 7]. This technique can be summarized as follows. For the reference pattern, code the boundary with respect to a fixed reference point. For the test pattern, use this coding to reconstruct the possible locations of the reference point. The possible locations are thus obtained. If the two patterns are identical, there would be a peak at the location of the original reference point.

In this form, the Hough transform method has several limitations. It requires the reference and test objects to be of the same scale and orientation. To account for orientation, the above procedures must be repeated for every orientation to be distinguished. Thus computational complexity increases rapidly if it is necessary to deal with variations in scale and orientation. A more serious objection is that the transform suffers from false peaks in the accumulator array due to random matches caused by noise or distortion.

In the Affine transform method, the same voting technique is used as that in the Hough transform method. However, it does not have the shortcomings such as translation, orientation and scale variant problems, or misrecognition due to false peaks. Therefore the Affine transform method is often preferred. The Affine transformation mathematically is an automorphism. An automorphism of a mathematical structure is always a one-to-one mapping of that structure onto itself which preserves its structural properties. Therefore, Affine geometry could be called the geometry of parallelism, that is, Affine geometry has the characteristics of parallelism and the preserving of the ratio of parallel line.

In object recognition terms, any two different top view images of the same flat object are in an Affine 2-D correspondence. The Affine transform method involves a nonsingular 2 x 2 matrix A and 2-D (translation) vector b, such that each point χ in the first image is translated to the corresponding point $A\chi + b$ in the second image. Our problem in this thesis is to find the identity of objects in the scene and the Affine transformation between their locations in the scene and the stored models. The study is concentrated on recognition of flat rigid objects. However, this method can be extended to general and 3D objects. More detail about the

Affine transformation method will be addressed in the following chapters.

In Chapter II there is a discussion of the basic concept of Affine invariant transform illustrated through translation, rotation and enlargement of an arbitrary incoming sensed pattern. Motivated by this concept, a model-based object recognition system was developed. Chapter III details the algorithm and the system implementation that consists of three general object recognition phases: *feature extraction (preprocessing)*, *object modeling* and *object matching (recognition)*. Chapter IV discusses possible ways to improve the system performance and discussion of topics for further study. The Pascal-like pseudo code programs are included in Appendix A to explain the implementation. Appendix B contains the entire Affine invariant object recognition system source codes.

II. AFFINE INVARIANT MATCHING

The occluded or overlapped object recognition problem has received more attention recently in the object recognition community. A recognition technique that can identify an occluded object is discussed in this chapter. To understand many of the concepts involved in this work, a basic background for the Affine transformation is presented. The application of Affine transformation in the object recognition through match and voting techniques is also discussed.

A. PARTIAL MATCHING

In the industrial robotic vision application, parts often appear to be occluded to the sensor. Occluded objects are difficult to recognize by using the traditional or global feature recognition method. Object recognition techniques using global feature need to know the complete information representing or describing the objects. Once some portions are covered by other objects, the information will cause the recognition to fail. Accordingly, it is necessary to describe our objects by a set of local features. This situation is applicable to the human vision system, which is also capable of recognizing the object in the presence of considerable

occlusion. The local features can be points, line segments, curve segments, borders, or other structures obtained from feature extraction.

Initially, attention is restricted to the use of special points of the boundary, which we denote as *interesting points* The point sets of the various objects are matched against the point set of the composite overlapping test object using a small number of corresponding points. To elaborate upon these views, one particular geometrical transformation with a unique mapping characteristic is used. That is the Affine coefficient invariant transformation discussed below.

B. AFFINE TRANSFORMATION AND VOTING TECHNIQUE

1. Affine Coefficient

The *interesting point* set can be transformed to an *Affine coefficient* pair (ξ,η). Any three non-collinear points can uniquely specify a plane. The rest of the points of that plane can be expressed in terms of these three points. This representation is in terms of the *Affine coefficient* pair (ξ,η).

The objects to be recognized in our system are assumed to be planar objects. The known objects are stored as models in the data-base in advance. Recognition of test object by comparing it to the models is based on two different sets of *Affine coefficients* which are obtained from the test object and the models respectively. Figure 1 shows this concept graphically.

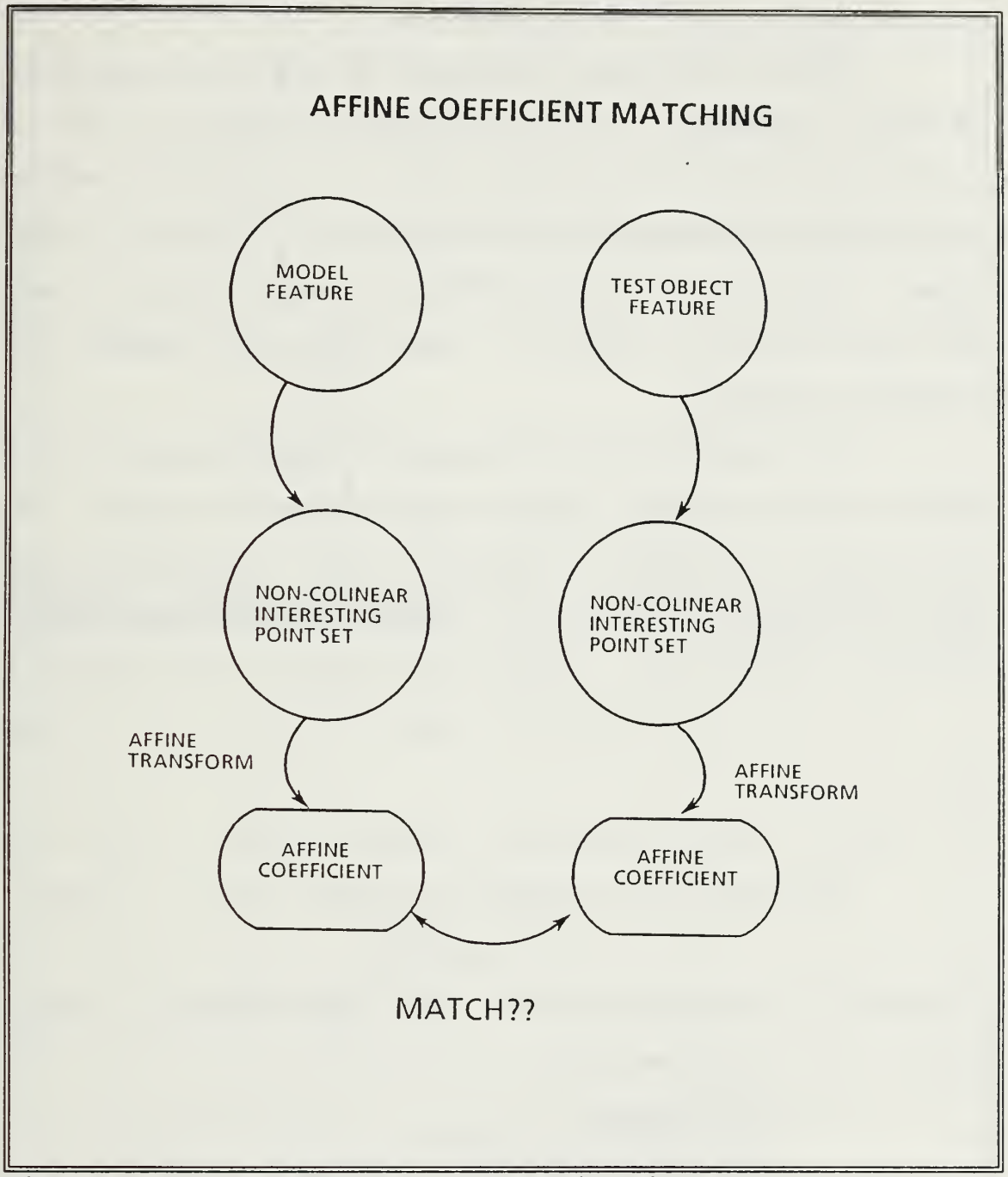


Figure 1. The Concept of Recognition in Affine Coefficient Space.

2. Affine Coefficient Transform

Affine coefficient matching is one of the important matching techniques. It is motivated by the fact that any Affine transformation of a plane [Ref. 8] is uniquely defined by knowledge of three non-collinear points (or triplet) in the plane. The Affine transformation is always unique; it maps any non-collinear triplet in one plane to another non-collinear triplet.

To illustrate the "invariant" characteristic of the Affine transformation, consider the following example. For the pattern of Figure 2, there are four vertices A=(0,0), B=(0,1), C=(1,0) and D=(1,0). Point D can be expressed by points A, B and C (i.e., triplet), through the representation.

$$D = \xi(B - A) + \eta(C - A) + A \tag{4}$$

i.e.,

$$(1,0) = \xi[(0,1) - (0,0)] + \eta[(1,2) - (0,0)] + (0,0) \tag{5}$$

Solving for the unknown variables ξ and η, we get

$$\xi = -2, \eta = 1$$

To show the characteristic of this transformation, let us examine the following cases.

a. Case I: Translation

Assume that there is a translation applied to the original pattern in Figure 2. The translation of the original pattern is accomplished by adding 1 to the coordinate of each vertices and results in Figure 3. In Figure 3, points A', B',

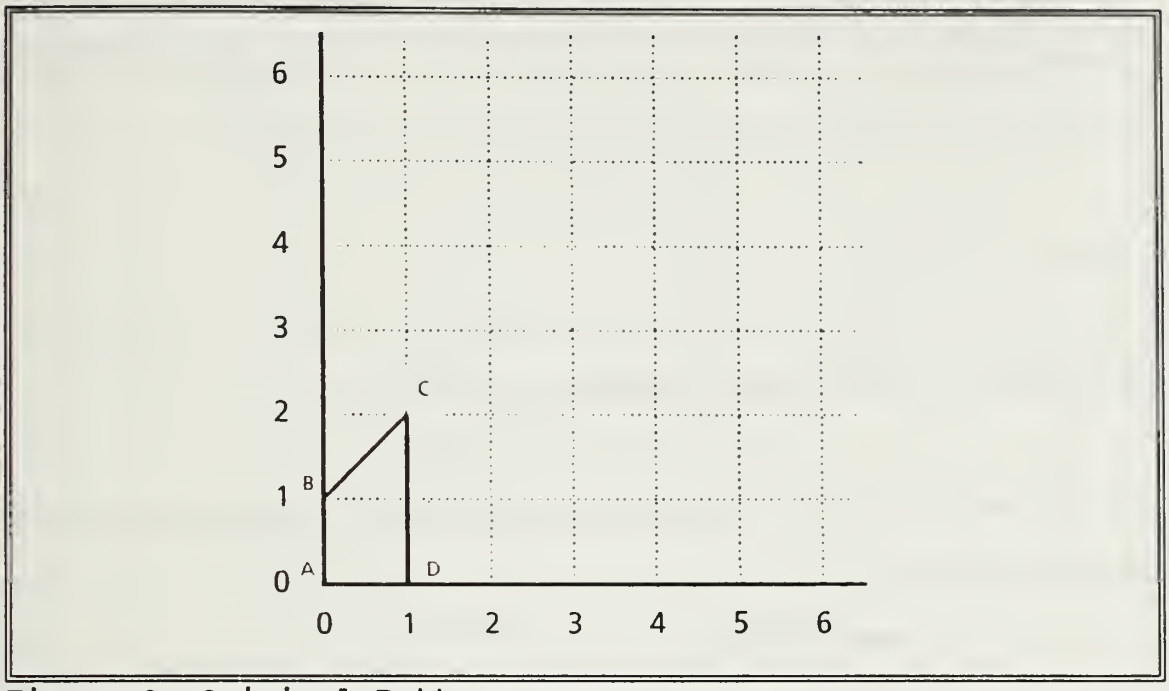


Figure 2. Original Pattern.

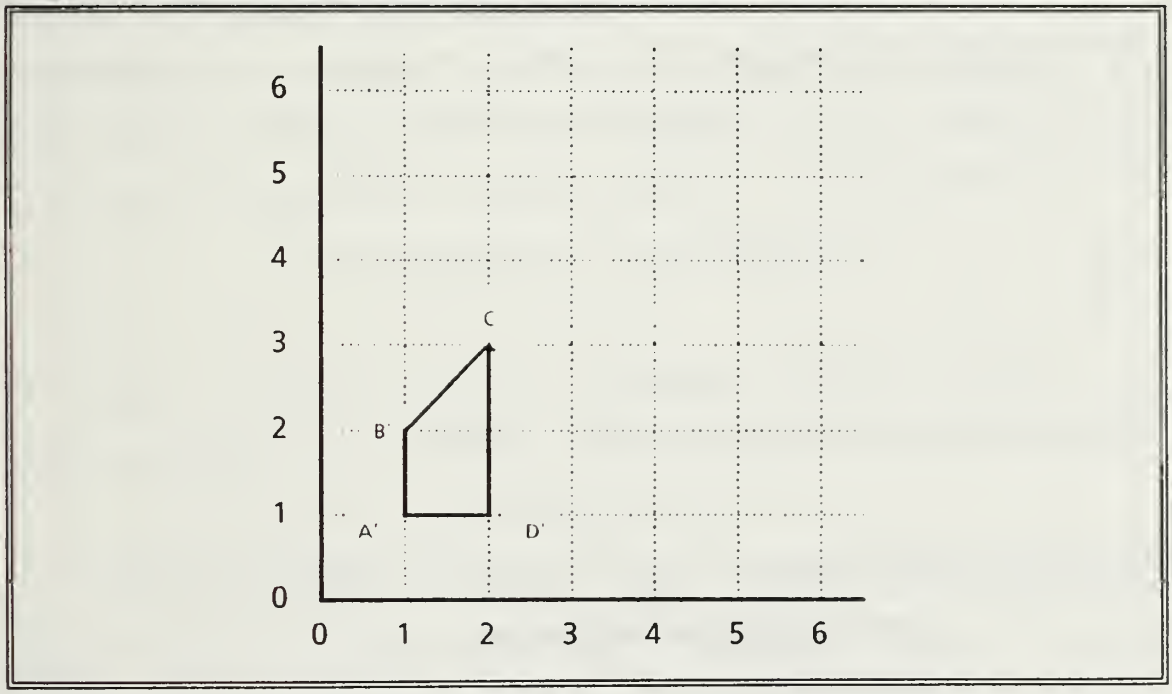


Figure 3. Translated Pattern.

C', and D' correspond to A, B, C and D respectively and have coordinates $A' = (1,1)$, $B' = (1,2)$, $C' = (2,3)$ and $D' = (2,1)$. Point D' can also be expressed by A', B' and C' as:

$$D' = \xi(B' - A') + \eta(C' - A') + A' \quad (6)$$

i.e.,

$$(2,1) = \xi[(1,2) - (1,1)] + \eta[(2,3) - (1,1)] + (1,1) \quad (7)$$

Solving for the unknown we again obtain

$$\xi = -2, \eta = 1$$

It is seen that the Affine coefficient pair remains the same in this case.

b. Case II: Rotation

Figure 4 shows the result of adding 1 and rotating the original pattern by 90°. Now, $A' = (2,3)$, $B' = (2,2)$, $C' = (1,1)$ and $D' = (1,3)$. Point D' can again be expressed in terms of A', B' and C' as:

$$D' = \xi(B' - A') + \eta(C' - A') + A' \quad (8)$$

i.e.,

$$(1,3) = \xi[(2,2) - (2,3)] + \eta[(1,1) - (2,3)] + (2,3) \quad (9)$$

Solving for the unknowns again, we get

$$\xi = -2, \eta = 1$$

The Affine coefficient pair is still the same as before.

c. Case III: Enlargement-Scale Change

The combined effect of translation, rotation, and enlargement, which is generated from the original pattern by adding 1 (translation), rotating 90°, and multiplying by 2

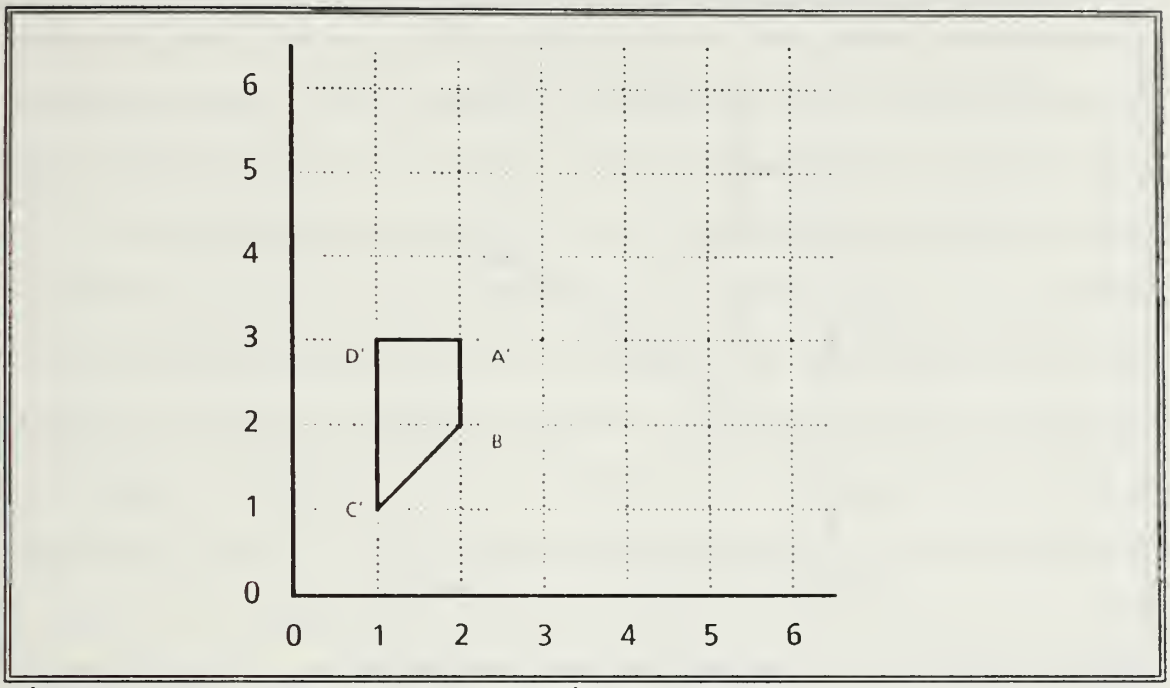


Figure 4. Pattern After Rotation.

(enlargement) is shown in Figure 5. Here, the vertices are $A' = (4,6)$, $B' = (4,4)$, $C' = (2,2)$ and $D' = (2,6)$. Point D' can also be expressed by A', B', and C' as:

$$D' = \xi(B' - A') + \eta(C' - A') + A' \tag{10}$$

i.e.,

$$(2,6) = \xi[(4,4) - (4,6)] + \eta[(2,2) - (4,6)] + (4,6) \tag{11}$$

Again we find

$$\xi = -2, \eta = 1$$

In all those four cases, ξ and η are exactly the same. We have illustrated the invariant property of the transformation in the above cases. That means that the Affine coefficient pair (ξ,η) does not change in those four cases. The Affine invariant matching technique studied in this thesis is based on this property.

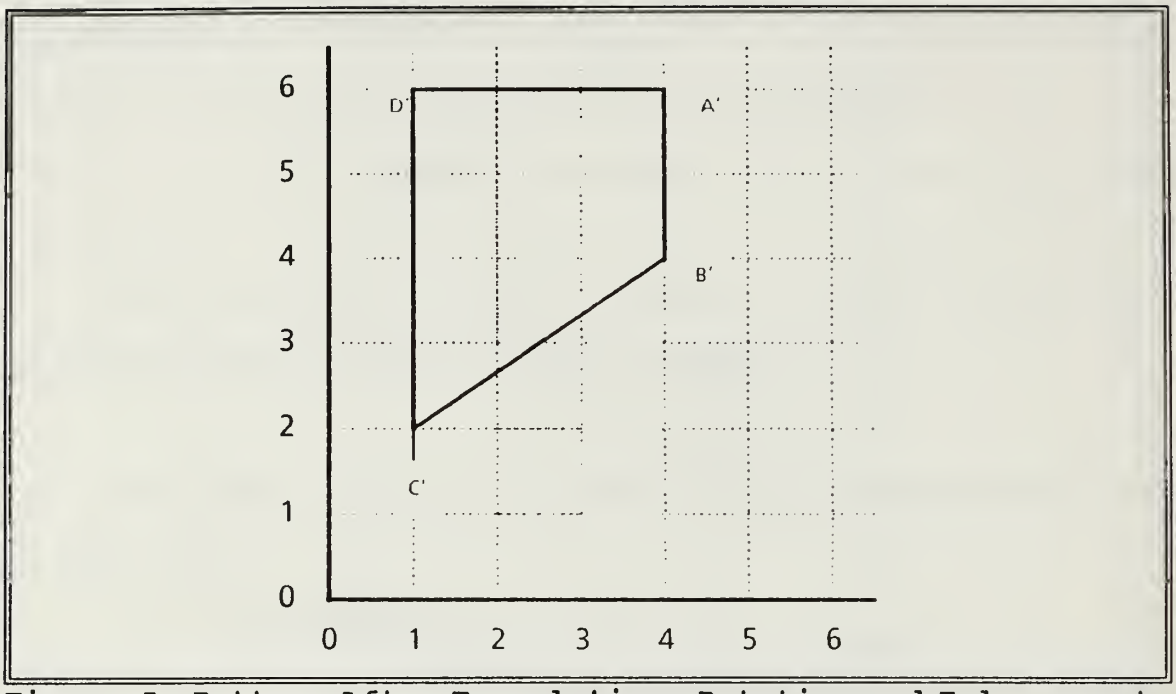


Figure 5. Pattern After Translation, Rotation and Enlargement.

For a test object in the scene, we can exploit this property to represent those points of the test object in terms of their Affine coefficients which are invariant to similarity transformations. That is, we change the object representation from the interesting point representation of the original plane (the higher dimension representation) to a representation based on the non-collinear triplet (the lower dimension representation). In this coefficient space, it is possible to apply search and voting techniques to recognize the test objects.

C. AFFINE INVARIANT RECOGNITION ALGORITHM

The Affine coefficient invariant property was illustrated in section II.B.2. It is possible to exploit this property

to recognize planar objects. We divide the recognition process into two separate procedures. The first procedure is *Modeling (encoding)*. The second procedure is the *Recognition (Matching)*. The *Modeling* procedure has two phases: *preprocessing* and *data-base setup*. The recognition procedure has three phases: *Preprocessing*, *Data-base access* and *Voting match.* The purpose of the *Preprocessing* is to extract a set of *interesting points* from the object. It is used both in the *Modeling* procedure and the *recognition* procedure. The *Date-base setup* includes choosing three arbitrary non-collinear points (triplet) as an Affine basis and generating the coefficients ξ and η. These are used as index keys to create a search table with the triplet as its content. In the *Recognition* procedure the Affine coefficient of the interesting points of the test object (ξ,η) is also used as the index keys to search the previously created table. When there is a match, we increment a vote. The total vote will be used to decide whether the basis of the test object is the one corresponding to that of the model.

1. Preprocessing (Feature Extraction)

This phase is done to extract features for either the models in the date-base or the test patterns. These features can be points, line segments, curve segments, etc., as mentioned in Chapter I. In general, the more features extracted, the more detail it is possible to describe a model. Hopefully, with more features, more accurate recognition can

be achieved. However, this approach will increase computation complexity. Using the extracted features, a model approximation is formed, using the set of interesting points for each object model. Since the image is assumed to be flat, only one set of interesting points is involved. For 3-D objects, it may be necessary to extract several sets of interesting points, one for each orthogonal projection of the object. Smoothing and data reduction are also two of the primary effects in feature extraction. The thesis work studied here will not emphasize the *feature extraction* phase. The processing steps will be discussed in more detail in section III.A.

2. Data-Base Setups (Modeling)

This phase is the second step in the *Modeling (encoding)* procedure where creation or update of the model data-base is accomplished. The objective of this phase is to encode models in the data-base and to create the search table. Each model object is represented by a set of local feature interesting points which were obtained from the *Preprocessing*. A total of *m interesting points* are extracted from the model, (i.e., the model points). These model points approximate the object in a special form. For each ordered non-collinear triplet of model points the coefficient (ξ, η) of all the rest of the $m-3$ model points are computed. Taking this triplet as an Affine basis in the 2-D plane, the other points can be represented as

$$\mathbf{v} = \xi(e_{10} - e_{00}) + \eta(e_{01} - e_{00}) + e_{00} \qquad (12)$$

where e_{00}, e_{01}, e_{10} are the three selected non-collinear points. Application of an Affine transformation T described in section II.B.2. will change the point $\mathbf{v}$ to

$$T\mathbf{v} = \xi(Te_{10} - Te_{00}) + \eta(Te_{10} - Te_{00}) + Te_{00} \qquad (13)$$

Hence, $T\mathbf{v}$ still has the same coefficient (ξ,η) on the new triplet basis Te_{00}, Te_{10}, Te_{01}. Each such coefficient is used as an index to access the search table. The entry of this search table may have a variable number of ordered pair, (*modelid*, *basis-tripletid*). The ordered pairs helps to access the vote array to register the basis-triplet by which the coefficient was obtained. Some detail of the search table is shown in Table 2. This is a conceptual table mainly used to illustrate the algorithm. The actual implementation is different in detail. That will be discussed in the next chapter.

3. Voting Match

In the recognition procedure a test object is given in a scene. First, the interesting points of this test object (say we have *n interesting points*) have to be extracted. An arbitrary ordered test triplet of the interesting points of the test object is selected. Based on the test triplet, the Affine coefficient of the other points can be calculated. The search

table consists of entries indexed by (ξ,η). Each entry may have a variable number of records of ordered pair (*modelid*, *basis-tripletid*), where the *model* field is an integer for model identification and the *basis-triplet* field is the model triplet used as the Affine basis for the model. Associate with each record of ordered pairs is a vote array called "vote." This is initialized to zero.

Table 2. CONCEPTUAL SEARCH TABLE FORMAT.

1ST AFFINE COEFFICIENT (KEY1,KEY2)	(MODELID,TRIPLETID)	VOTE	(MODELID,TRIPLETID)	VOTE	....
2ND AFFINE COEFFICIENT (KEY1,KEY2)	(MODELID,TRIPLETID)	VOTE	(MODELID,TRIPLETID)	VOTE	...
3RD AFFINE COEFFICIENT (KEY1,KEY2)	(MODELID,TRIPLETID)	VOTE	(MODELID,TRIPLETID)	VOTE	...
...	...	...	...	...	...

During the recognition procedure, for each Affine coefficient pair (ξ,η) the search table created in the *Data-Base Setup* phase is accessed and those vote fields are also tallied. For each entry of the table, different (*modelid*, *basis-tripletid*) pairs with the same (ξ,η) are accessed and the entries of "vote" are incremented by one. If the entry of the search table scores a large enough number of votes, all ordered pairs of this entry become possible candidates to match the test

object. The next step is to decide which of the ordered pairs (i.e., which model) will be the real matched one. The selection process is based on the fact that the model with the maximum accumulated vote is the matched model. The model triplet of the ordered pair of the maximum accumulated vote then corresponds to the one chosen for the test object. The uniquely identified Affine transformation, between the model triplet and the test triplet, is the selected transformation between the model and the object. Using the selected model triplet and all entry of the pairs (ξ,η) associated with this selected maximum vote model triplet, it is possible to reconstruct the recognized model in the scene. If the selected model triplet and all of the associated pairs (ξ,η) do not match the dimension of the model, the second maximum vote model triplet and its associated pair (ξ,η) is checked in the same way. Because of this checking procedure the test object can be reconstructed even when there is occlusion.

III. ALGORITHM IMPLEMENTATION

A. SEARCHING

Searching plays an important role in the Affine recognition procedure. It is necessary to search the table shown in Table 2. Search is dependent on the index or search key. This is a content dependent search. The objective of the search is to collect the instances (*basis-triplet*) of the models which occurred in the test object. The more instances of the model occurring in the test object, the more confident will be the decision which shows what part of the test object resembles the model. Content dependent search is inherently slow. Therefore, special techniques are needed to speed up this procedure.

B. HASH TECHNIQUE

In the recognition procedure, the result of the Affine transformation of the interesting points is actually a pair of real coefficient, i.e., (ξ,η). By multiplying it with a constant, this pair of coefficients can be added to generate the key, i.e., *inkey*, in this system. Since the quantized real coefficient is somehow random, it is not desirable to use sequential array data structure like those shown in Table 2.

The storage overhead and speed reduction will be too severe. The Hashing technique is a preferred technique. The *inkey* is associated with a location of an *accumulator cell*. The *accumulator cell* is used to store the instance of the occurrence of the candidacy model.

Hashing is a many-to-one mapping that does not usually preserve the order of the keys of a list. Hashing is achieved by deriving a *hash key* from the *inkey* which indicates to which sublist or neighborhood a record belongs. This sublist is called a *bucket*. The biggest problem is Hashing is the difficulty in producing a relatively uniform number of records in the *bucket*. When the keys, *inkey1* and *inkey2*, of two distinct records are converted to the same *hash key*, there is a collision. To solve this problem in this system, it is necessary to use a linked list to append the collided record after the last record in the *bucket*. The linked list is used to link all records in the same *bucket* together with the same *hash key*.

There are several possible Hashing implementations. One of the best general purpose methods is the division method [Ref. 9]. Our system is based on this method. In this method an *inkey* is divided by a number which is the table size in this system. Then, the record is assigned to the *bucket* that is associated with the remainder, i.e., the *hash key*. Table size is determined by the possible number of (ξ,η) pairs generated

from the interesting points of the model for all bases. Figure 6 shows *hash key* generation by division.

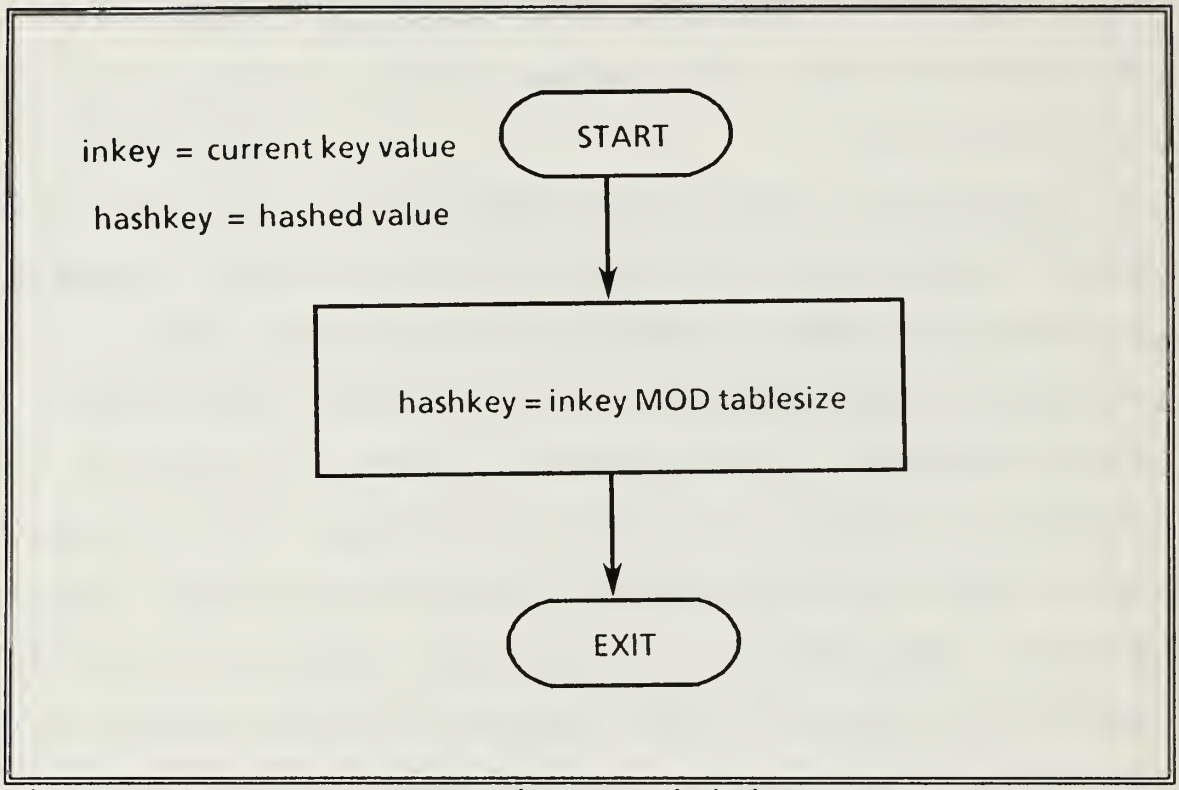


Figure 6. Hash Key Generation by Division from *Inkey*.

1. Hashing Implementation

The algorithm implementation is divided into two separate procedures. The first procedure is *Modeling (encoding)*. The second procedure is *Recognition (Matching)*. The *Modeling* procedure has two phases: *preprocessing* and *data-base setup*. The recognition procedure has three phases: *preprocessing*, *data base access*, and *voting match*. The implementations for each of these modules are described in sequence. In *preprocessing* the interesting points of the boundary are formed. In *data base setup* the Affine coefficient pairs (ξ, η) are calculated which are

used to generate the "Hash" table. In *voting match* the test object and models in the data base are compared in terms of (*modelid*, *basis-tripletid*) pair.

C. PREPROCESSING

The preprocessing is shown in Figure 7, which consists of a number of steps. All the modules in this phase are adopted from the commercial package SPIDER [Ref. 10] on a VAX 11/780 system. SPIDER is an image processing utility software package. There are five routines written in FORTRAN used for preprocessing.

1. Laplacian Operator EGLP [Ref. 10]

The Laplacian function f(X,Y) of two independent variables is defined as

$$\nabla^2 f(X,Y) = \frac{\partial^2 f(X,Y)}{\partial X^2} + \frac{\partial^2 f(X,Y)}{\partial Y^2} \tag{14}$$

The image intensities change sharply at places where $\nabla^2 f(X,Y)$ is large. The outline is detected and obtained from the Laplacian image plane.

2. Threshold Operation SLTH

This operation thresholds the output data of the Laplacian operator and generates the input data for the boundary tracing operation conducted later. That is,

$$g(x,y) = \begin{cases} 256 & \text{if } \nabla^2 f(X,Y) > THRESHOLD \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

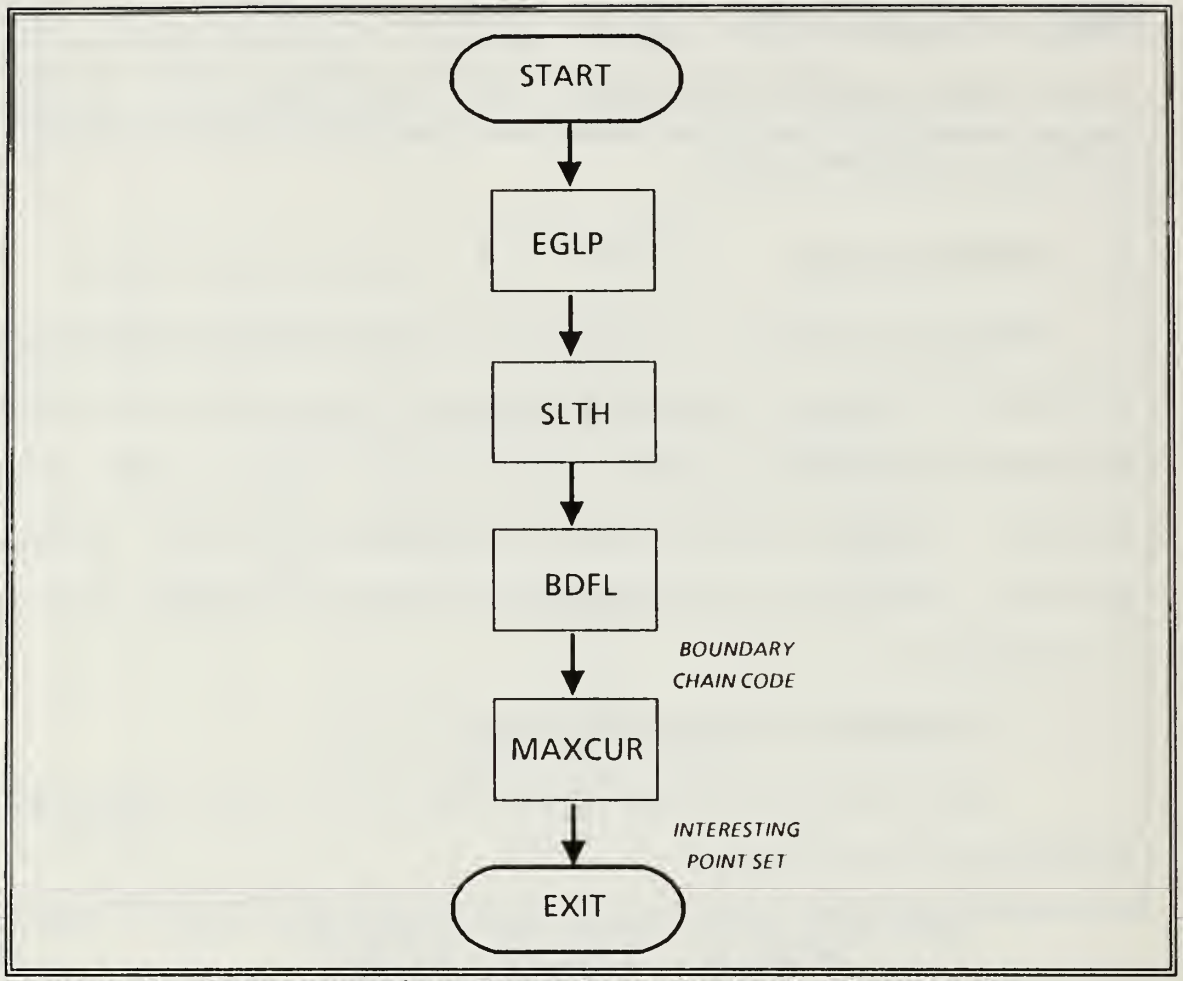


Figure 7. Preprocessing Procedure Control Flowchart.

3. Boundary Thinning THNG [Ref. 10]

The boundary outline of the image is thinned to a width of 3 pixels.

4. Tracing Boundary BDFL [Ref. 10]

Boundary tracing is the conversion from boundary coordinates to chain codes [Ref. 11]. The line is represented by the coordinates of the starting point and a series of codes indicating the slope of the line segments (or links). The

BDFL algorithm primarily involves raster scan and tracking of the data.

5. Maximum Curvature Finding MAXCUR

The interesting points are taken at places where the chain code slope is greater then a certain threshold.

At the end of the preprocessing phase, the interesting points of the boundary are selected from either the model or the test object.

D. DATA-BASE SETUP

The *data-base setup* phase is shown in Figure 8. In the *data-base setup* phase of the *modeling* or *encoding* procedure, two steps are involved. They are discussed in the following.

1. Affine Transformation

AFFINE is the first step in the *data-base setup* phase. From the interesting points, a set of non-collinear triplets is chosen as an Affine basis. Then the Affine coefficient pair (ξ,η) of the other points are calculated. The Affine coefficient pair (ξ,η) is then converted from real data type to integer type by multiplying it by a constant. For example, if there are four interesting points (X1,Y1), (X2,Y2), (X3,Y3) and (X4,Y4), the fourth point is expressed using the other three as a triplet basis,

$$X_4 = \xi(X_1 - X_2) + \eta(X_2 - X_3) + X_3 \qquad (16)$$

$$Y_4 = \xi(Y_1 - Y_2) + \eta(Y_2 - Y_3) + Y_3 \qquad (17)$$

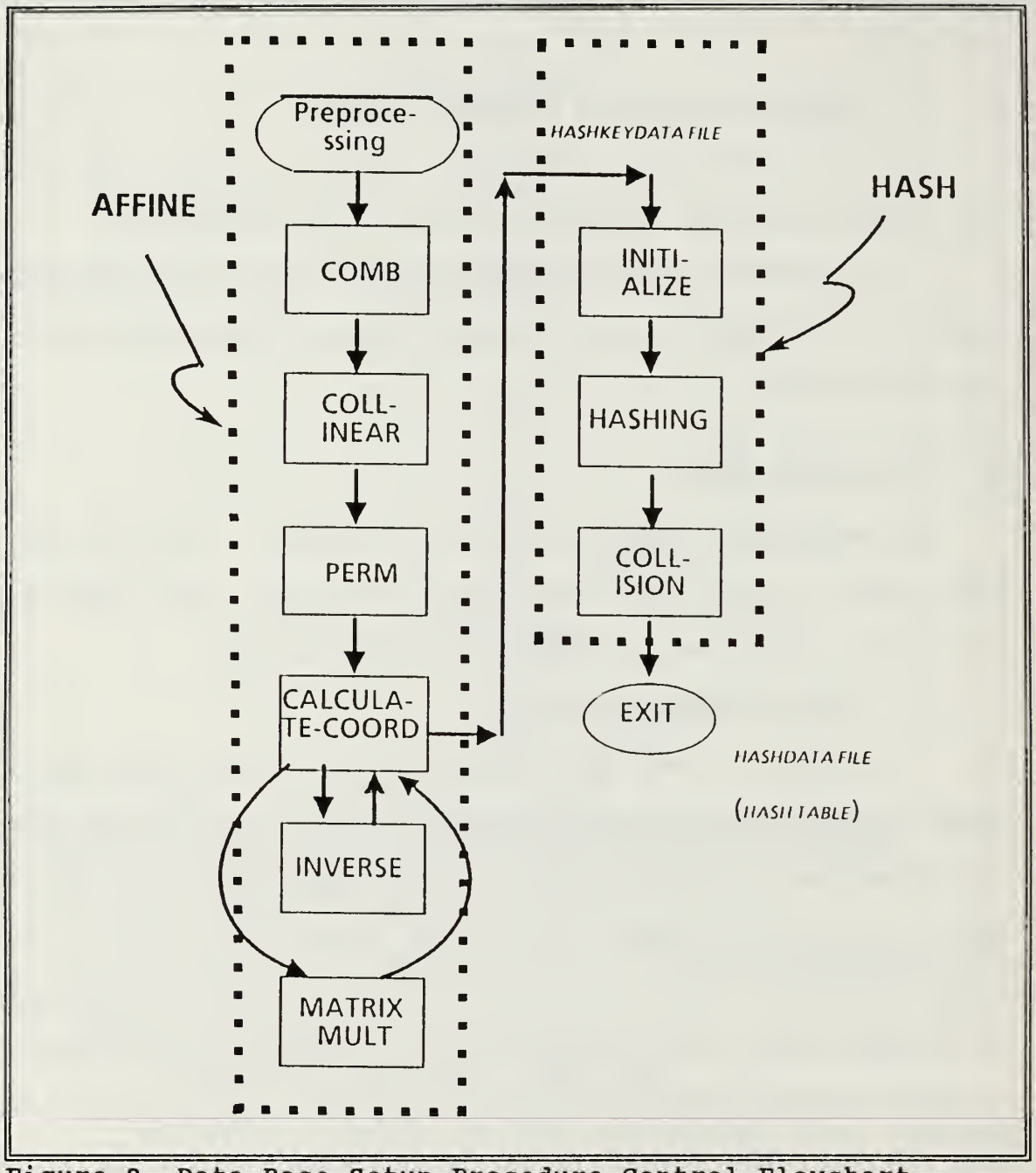


Figure 8. Data-Base Setup Procedure Control Flowchart.

and ξ and η are solved from the above equations. That is

$$\begin{pmatrix} \xi \\ \eta \end{pmatrix} = \begin{pmatrix} (X_1 - X_2)(X_2 - X_3) \\ (Y_1 - Y_2)(Y_2 - Y_3) \end{pmatrix}^{-1} \begin{pmatrix} (X_4 - X_3) \\ (Y_4 - Y_3) \end{pmatrix} \qquad (18)$$

This procedure is repeated for all different ordered triplets (i.e., permutations and combinations) as basis until every possibility is exhausted. This will generate an output file (*hashkeydata*). (The format of this data file is described in Table 3.) The main Affine program reads the interesting point data from the input file *interestingdata* and invokes the following modules to accomplish the task. The Pascal-like pseudo code and source program of all modules are listed in Appendix A.

Table 3. HASHKEY DATA FILE FORMAT.

	inkey	model id	triplet id	key 1	key 2	model triplet base X1,X2,X3	model triplet base Y1,Y2,Y3
entry 1	-20000000	1	1	-1	-1	30,20,40	40,30,30
entry 2	0	1	1	1	-1	30,20,40	40,30,30
.....	....	....	....	....	....	....	....
entry i	10000000	2	1	1.5	-0.5	40,30,20	10,10,30
.....	....	....	....	....	....	....	....
entry m	....	....	....	....	....	....	....

Note : *inkey* = 10000000* (*key 1* + *key2*)
entry1 and entry 2 belong to the same model triplet

a. COMB Module

This module is a recursive module which generates all the possible combinations of triplets from the input interesting points. If there are *m* interesting points, the total number of combinations will be

$$ {}_mC_3 = \frac{m(m-1)(m-2)}{3!} \qquad (19) $$

For each combination this module will call the COLLINEAR module.

b. COLLINEAR Module

This module tests the collinearity of the combinations sent from the *COMB* module. Slopes among the points are checked to verify the collinearity. If the test of collinearity is not successful, this module then calls the *PERM* module. If the test is successful, control will be returned to the *COMB* module for a new combination.

c. PERM Module

Given the combination passed from the *COLLINEAR* module, this module generates all the permutation. This is again a recursive program. For each permutation it calls the *CALCULATE-COORD* module.

d. CALCULATE-COORD

This module calculates the *base-matrix*

$$base - matrix = \begin{pmatrix} (X_1 - X_2)(X_2 - X_3) \\ (Y_1 - Y_2)(Y_2 - Y_3) \end{pmatrix} \tag{20}$$

It then calls the *INVERSE* module by passing the *base-matrix* for inversion. Then it calculates the difference for the rest of the interesting points (X_i, Y_i)

$$difference = \begin{pmatrix} (X_i - X_3) \\ (Y_i - Y_3) \end{pmatrix} \tag{21}$$

This module then invokes the *MATRIX-MULT* module to get the final solution, the Affine coefficient pair (ξ_i, η_i).

e. INVERSE Module

This module calculates the *inverse-matrix*, for the solution of ξ and η, that is

$$inverse-matrix = \begin{pmatrix} (X_1 - X_2)(X_2 - X_3) \\ (Y_1 - Y_2)(Y_2 - Y_3) \end{pmatrix}^{-1} = \frac{1}{det} \begin{pmatrix} (Y_2 - Y_3)(X_2 - X_3) \\ (Y_1 - Y_2)(X_1 - X_2) \end{pmatrix} \quad (22)$$

f. MATRIX-MULT Module

This module calculates the final Affine coefficient pair (ξ_i, η_i). That is

$$\begin{pmatrix} \xi_i \\ \eta_i \end{pmatrix} = \begin{pmatrix} (X_1 - X_2)(X_2 - X_3) \\ (Y_1 - Y_2)(Y_2 - Y_3) \end{pmatrix}^{-1} \begin{pmatrix} (X_i - X_3) \\ (Y_i - Y_3) \end{pmatrix} \quad (23)$$

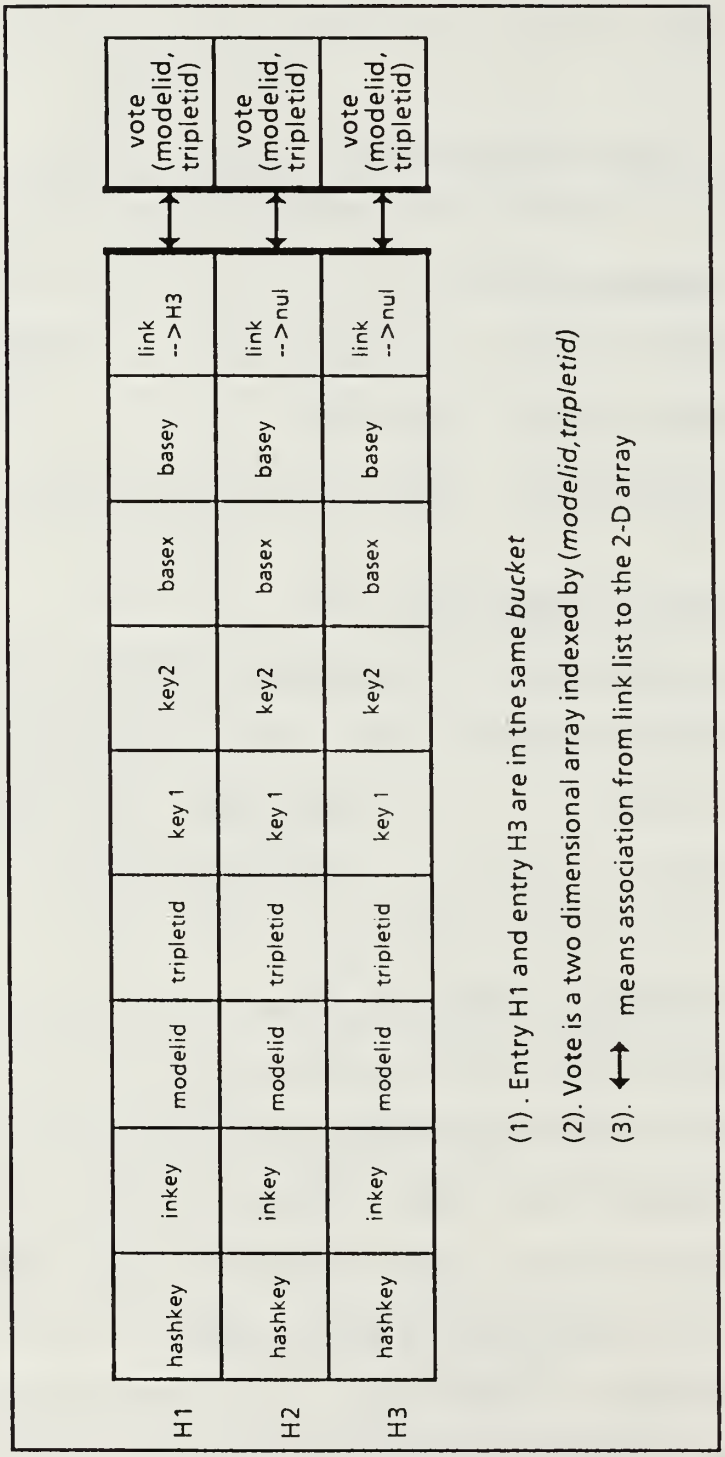
2. Hash Table Generation

HASH is the second step in the *Data base setup* phase. At the end of AFFINE execution, the (*modelid*, *basis-tripletid*) pairs are created and stored in a file (*hashkeydata*) with other support information shown in Table 3. This is an input file to the *HASH* module.

The purpose of the HASH program is to read the input file hashkeydata and create the Hash table with the format as shown in Table 4. The Hash table is a linked list structure.

Each record has extra fields: *hashkey*, *inkey*, and *link*. This structure is the actual implementation used in the experiment; it is different from the search table in Table 2 used for conceptual illustration. The *link* field is used to link the *collided* records while the *Hash* module maps two records with two different values of *inkey* to the same *hashkey*. As an

Table 4. THE HASH TABLE FORMAT.



example, for a module with four interesting points, the Hash table size is 4 x 6 = 24 where 4 is the combinations of three out of four and 6 is the permutations of three out of three. The main task of the *HASH* module is to generate a consistent Hash table for the recognition procedure. Three modules are called from the *HASH* program.

a. INITIALIZE Module

This is called to initialize the field *hashkey* and *link* all of the record to empty label.

b. HASHING Module

After initialization, the *Hash* main program invokes the *HASHING* module to convert Affine coefficient pair (ξ,η) (i.e., (*key1*,*key2*)) to *inkey* by adding them and multiplying the sum with a constant *multiplier*. That is:

$$inkey = (key1 + key2) \times multiplier \qquad (24)$$

The *HASHING* module also generates the *hashkey* as following:

$$hashkey = inkey \ \mathrm{MOD} \ tablesize \qquad (25)$$

The *HASHING* module then tests if the record accessed by the *hashkey* in the Hash table has an empty *hashkey* field. If so, it will insert the current record of the input file (*hashkeydata*) into the Hash table. Otherwise it will invoke the *collision* module.

c. *COLLISION Module*

This module tests to see if the accessed record's *link* field of the Hash table is empty. If it is empty, the current record of the input file is the first new-comer into the *bucket* list pointed to by the *link* field. The current record is put in the next available position in the Hash table. The *link* field of the accessed record is updated to point to this new position. If the *link* field is not empty, then using the *link* field, the module can find sequentially the last existing entry of the bucket list and append the current record after the last entry.

E. RECOGNITION

Once the *Modeling* (*encoding*) procedure is accomplished, it is possible to start the recognition procedure of a test object. The flow chart of recognition procedure is shown in Figure 9.

After the preprocessing, a set of n interesting points of the input test object were generated. The *AFFINE* program is invoked to process these points. This program includes modules *COMBINATION*, *COLLINEAR*, *CALCULATE-COORD*, *INVERSE* and *MATRIX-MULT* to obtain the Affine coefficient pairs (ξ_i, η_i) $i = 1,2,\ldots,n-3$ of the test object. Those Affine coefficient pairs (ξ_i, η_i) and the associated test triplet are stored in an output file (*candidatedata*). The format of output file (*candidatedata*) is shown in Table 5. *Inkey* is used to access the

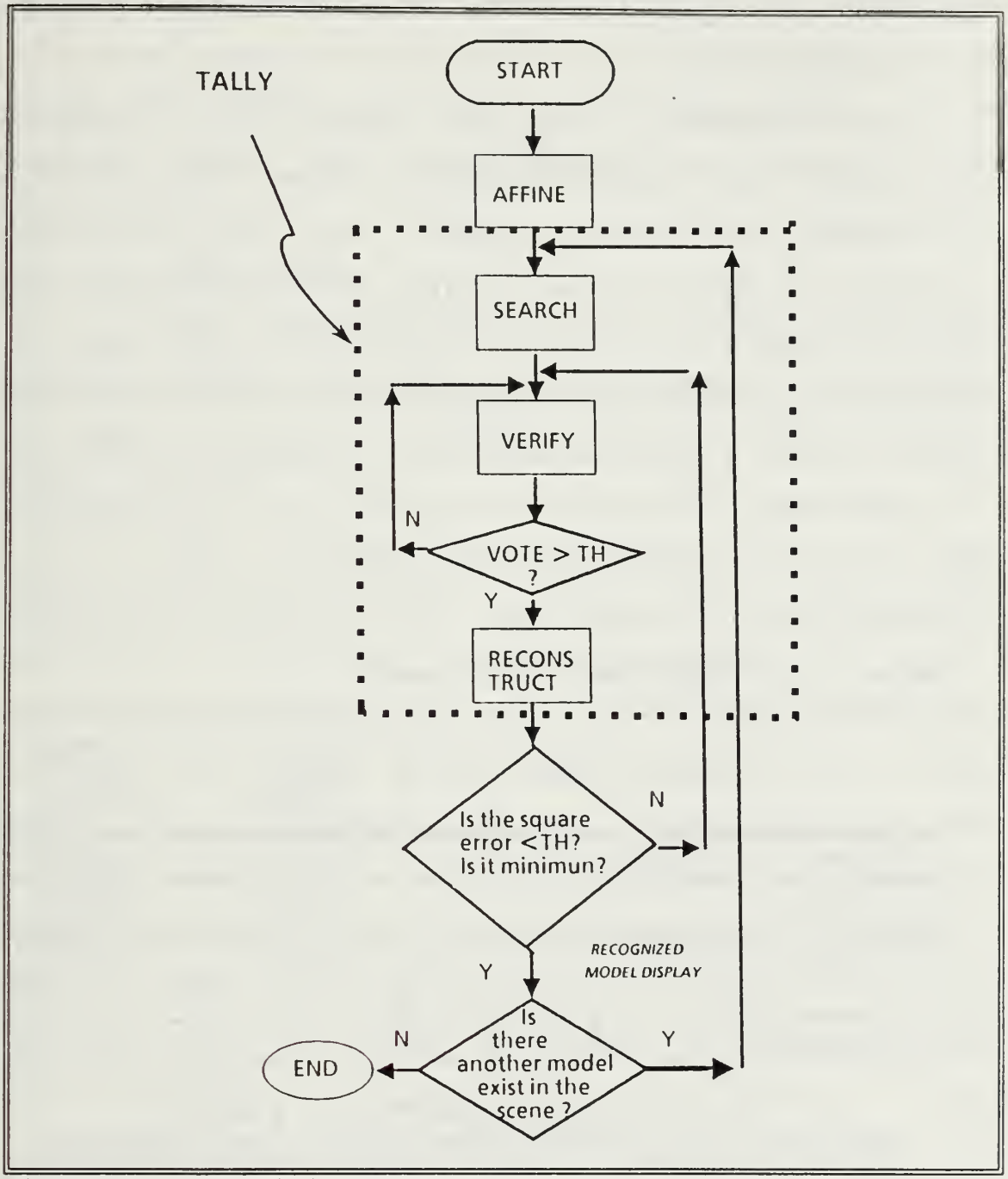


Figure 9. Recognition Procedure Flowchart.

Hash table and the test triplet is used to establish the correspondence between the model triplet of the Hash table,

and the test object at the end of the recognition procedure for reconstruction.

The objective of the *tally* program is to accumulate incidences of correspondence between one test object's triplet (*candidatedata*) and one model triplet. Each model has a set of triplets and there could be multiple models in the Hash table. Both of these two set of triplets and the associated interesting points could generate the same Affine coefficient pairs (ξ_i, η_i). This represents one incidence of the match. In other words, the Affine coefficient pair (ξ_i, η_i) generated by a test triplet is used to access the Hash table. Matching between test triplet and model triplet is done in the Affine

Table 5. CANDIDATE DATA FILE FORMAT.

	inkey	key 1	key 2	test triplet base X X1,X2,X3	test triplet base Y Y1,Y2,Y3
candidate 1	-20000000	-1	-1	30,20,40	40,30,30
candidate 2	0	1	-1	30,20,40	40,30,30
candidate 3	10000000	1.5	-0.5	40,30,20	10,10,30
.....	....	....	....	....	....
candidate n	....	....	....	....	....

Note : *inkey* = 10000000 * (*key 1* + *key 2*)
candidate 1 and candidate 2 belong to the same test triplet

coefficient domain. During the search, whenever there is a corresponding model triplet in the Hash table with the same Affine coefficient pairs as those of the test triplet, this model triplet in the Hash table is declared to match the test

object. The program *tally* first reads the input file (*candidatedata*) of the test object into a record array (*cand*) and then the hash table data (*hashdata*) into the record array (*model*). Three modules: *SEARCH*, *VERIFY*, and *RECONSTRUCT* are involved in this program.

1. SEARCH Module

Given a particular test triplet record from the input file (*candidatedata*), the Affine coefficient pair (ξ_i, η_i) and *inkey* are used to perform the access and comparison of the Hash table. That is

$$hashkey = inkey \text{ MOD } tablesize \tag{26}$$

Using the calculated *hashkey*, a certain entry of the created Hash table is accessed. If there is a match, i.e., a hit occurred in the Hash table, the corresponding 2-dimensional array *vote*(*modelid*, *tripletid*) shown in Table 4 with the same (*modelid*, *tripletid*) pair is incremented by one. At the same time, the accessed entry of the Hash table is moved to a temporary table. The data format of the *temporary* table is shown in Table 6. This table is needed later to reconstruct the model outline in the test object for graphic presentation. This entire procedure is executed iteratively until all of the interesting points of the test object based on the same test triplet are exhausted.

2. VERIFY Module

This module examines the 2-D array *vote*(*modelid*, *tripletid*) to select all the model triplets whose accumulated votes are all greater than a given threshold value. Based on the position (*modelid*, *tripletid*) of those that are greater than the threshold value, the model triplets are selected as the one most possibly corresponding to the test object's triplet.

Table 6. TEMPORARY TABLE DATA FORMAT.

	hashkey	inkey	model id	triplet id	key 1	key 2	accessed model triplet
temp 1	24	-20000000	1	1	-1	-1	
temp2	72	0	1	1	1	-1	
.....	.....	....	....	....	....	....	....
temp i	i	10000000	2	1	1.5	-0.5	

"i" is the number of the number of the test triplet

With the selected model triplets, control goes back to the temporary table to collect all those Affine coefficient pairs (ξ_i, η_i) for the selected model triplets. Comparison between the (*modelid*, *tripletid*) in the temporary table and the selected (*modelid*, *tripletid*) are necessary in this process.

3. RECONSTRUCT Module

The main purpose of this module is to reconstruct the model outline in the test object and check the squared error and for graphics presentation. Based on the selected model triplets and the collected Affine coefficient pair (ξ_i, η_i) from the *VERIFY* module, an inverse Affine transformation is performed

to calculate the sets of model coordinates in the model space. With the interesting point data file of the model the order of these points can be established. With the same (ξ_i, η_i) and the test triplet, the corresponding interesting points in the test space can be found according to the established order. The square error between each set of the model coordinates and the corresponding interesting points in the test space is measured. If the calculated square error is minimum and it is less than a given threshold, the optimum model triplet can then be determined. Based on the optimum model triplet, the graphics presentation of the model can be displayed in the test scene. In order to recognize all the possible models existing in the test object, after one model is recognized successfully, control goes back to the *SEARCH* module to find another peak in the array which has a different *modelid* and the rest of the procedure is repeated. If none of the calculated square error satisfies the given threshold, control goes back to the *VERIFY* module to use the second test triplet and repeat the procedure.

IV. RESULTS AND PERFORMANCE

A. EXPERIMENT RESULTS

The Affine invariant matching technique via hashing has demonstrated its characteristics of object recognition in several experiments. To concentrate the effort on the study of the Affine invariant technique and the implementation of the technique in software, the model data and test data were artificially created in the experiment. Two different models shown in Figure 10 were stored in the data-base after preprocessing phase.

1. Test For Similarity Transformation

The first unknown test object is fed into the system. The test object and the recognized model are shown in Figure 11. This is a case of recognizing objects under similarity transformations, i.e., rotation, translation and scaling.

The Affine invariant matching is obviously successful in detecting objects under a similarity transformation. Similarity transformation is a special case of a general Affine transformation. The key observation here is that since the similarity transformation is orthogonal, two points are enough to form a basis which spans the 2-D plane. A similarity transformation occurs in the situation when the

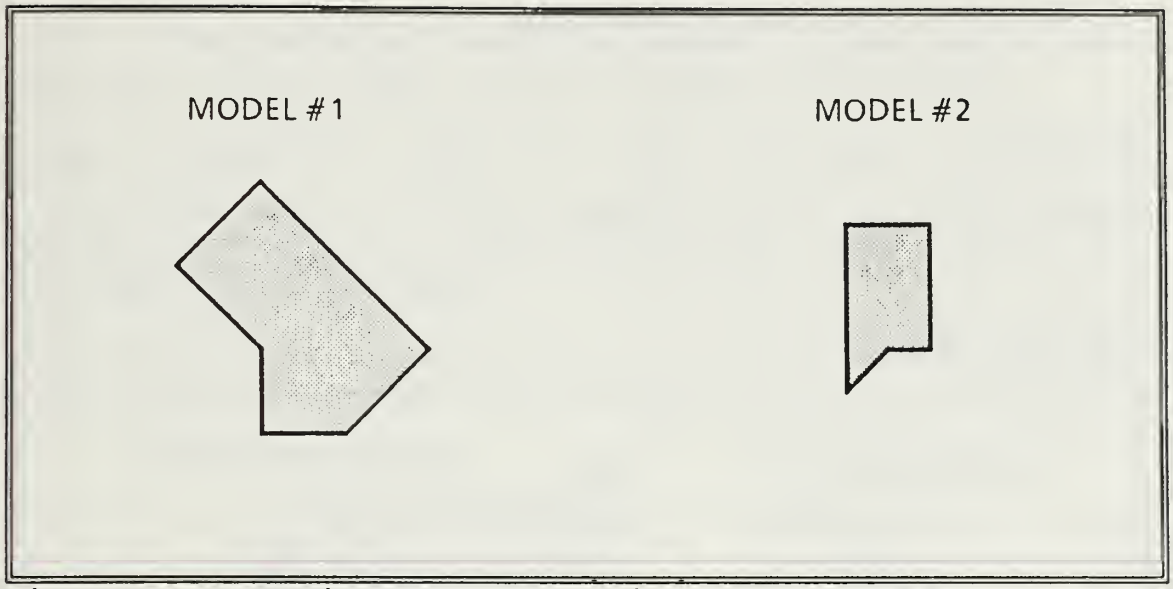


Figure 10. Two Different Models in the Data-Base.

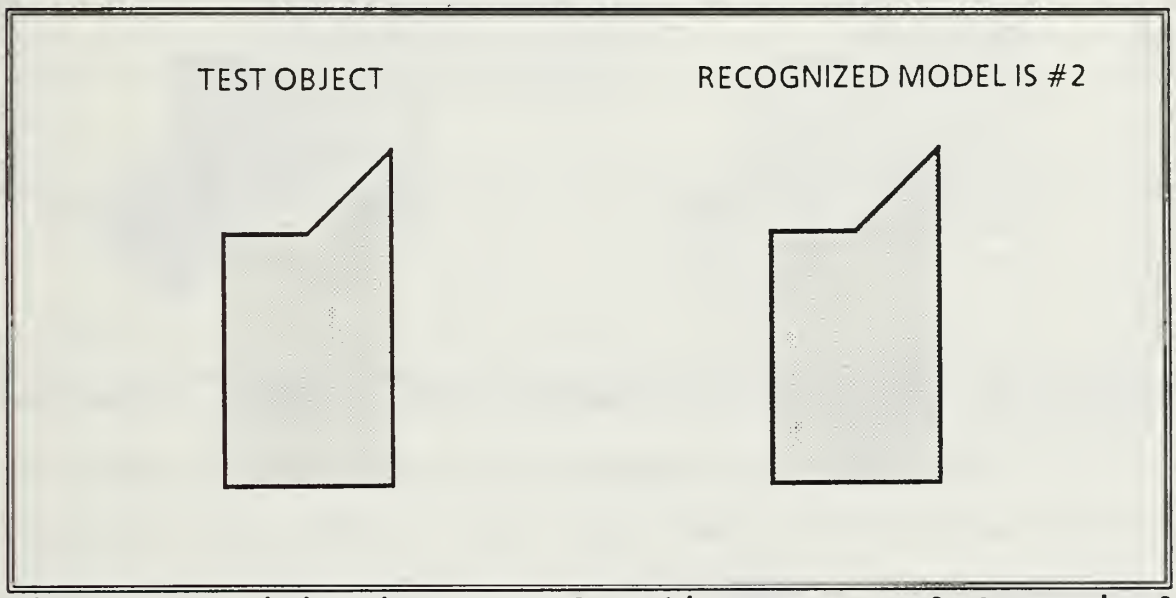


Figure 11. Similarity Transformation Case and Recognized Model.

viewing angle of the camera is the same as that of the model. Such a condition usually occurs, for example, in a factory environment where the viewing angle of the camera is kept constant. Consequently, this algorithm can be directly applied to industrial machine vision.

2. Test For Partial Matching

To test the situation where model can be identified partially, a test object of a composite overlapping scene in Figure 12 of the two different models from the data base is used. The recognition results are shown on the right hand side of Figure 12.

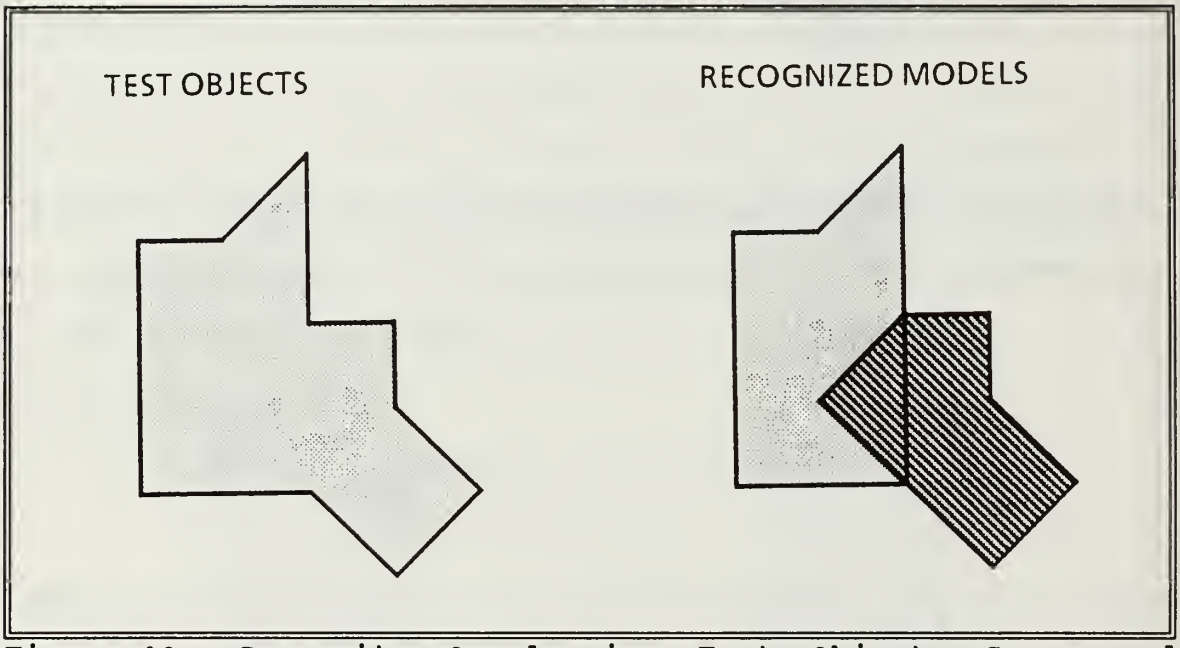


Figure 12. Composite Overlapping Test Objects Scene and Recognized Model.

Note that the hidden model's interesting point can still be reconstructed due to the partial matching characteristic of the algorithm. In other words, the hidden local feature such as the basis-triplet in our system can be recovered or assisted by the rest of the basis-triplets during the recognition phase.

3. When Numerical Error is Present in the Test Object

If the coordinates of *c* and *d* are very close to each other as shown in Figure 13, *c* and *d* cannot be part of the qualified triplets. The collinearity check module will discriminate these cases.

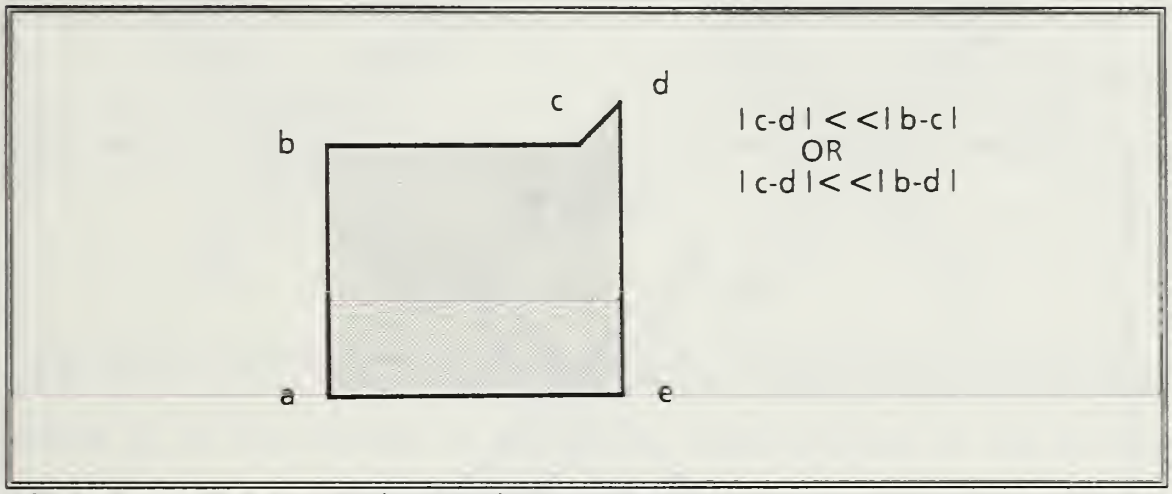


Figure 13. Interesting Point Too Close Case.

If the difference of *c* and *d* is significant, the triplet with *c* and *d* may result. A model triplet like this may get a number of votes, which on one hand, are not enough to accept it as a "selected" model triplet, but, on the other hand, do not justify total rejection. It is possible to change this model triplet to another model triplet consisting of points that are more distant from each other than those of the previous model triplet points. Even if a model triplet belonging to some model did not get enough votes due to noisy data, it is still possible to recover this model from another model triplet.

B. ALGORITHM PERFORMANCE AND COMPLEXITY

The performance is highly dependent on the number of interesting points m existing in the test scene. This is true as the data-base model is changed to that shown in Figure 14. The test object scene is shown in the left part of Figure 15 and the recognized model scene is shown on the right part of Figure 15.

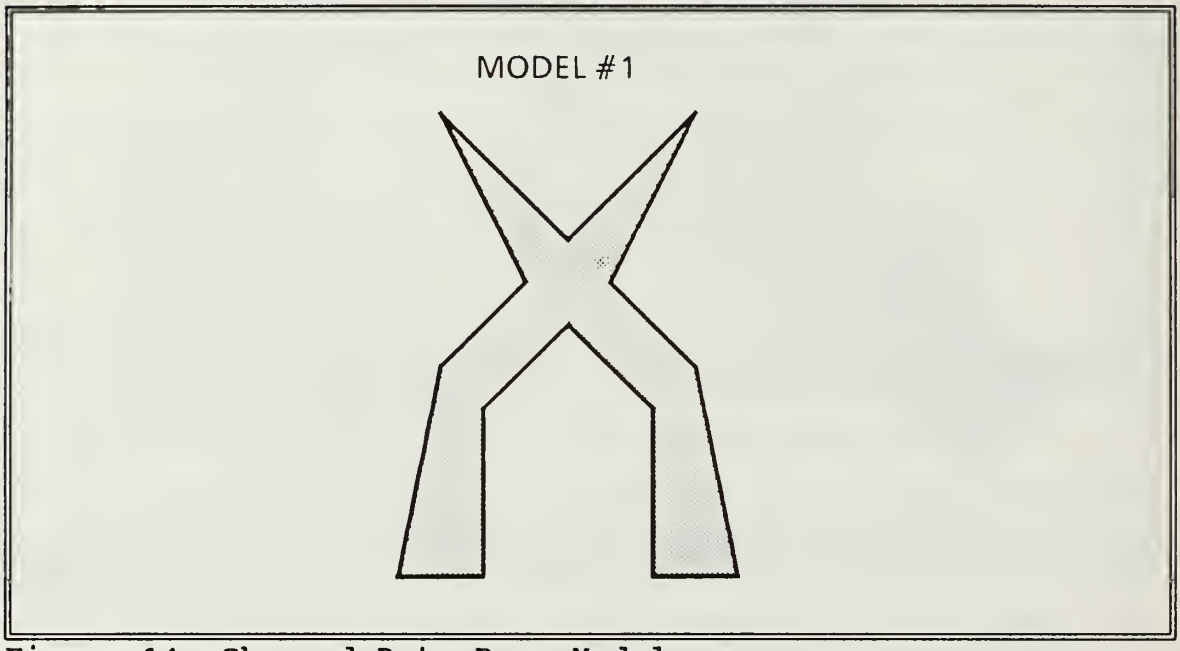


Figure 14. Changed Data-Base Model.

In this case the cpu time for recognition is increased tremendously. The total number of computation steps for m interesting points in Affine invariant matching is proportional to m^4 for the worst case. The Hash table size of order m^4 was obtained from

$$m \times {}_mC_3 \times {}_3P_3 = 6 \times m \times (m-1)(m-2)(m-3) \approx m^4 \qquad (27)$$

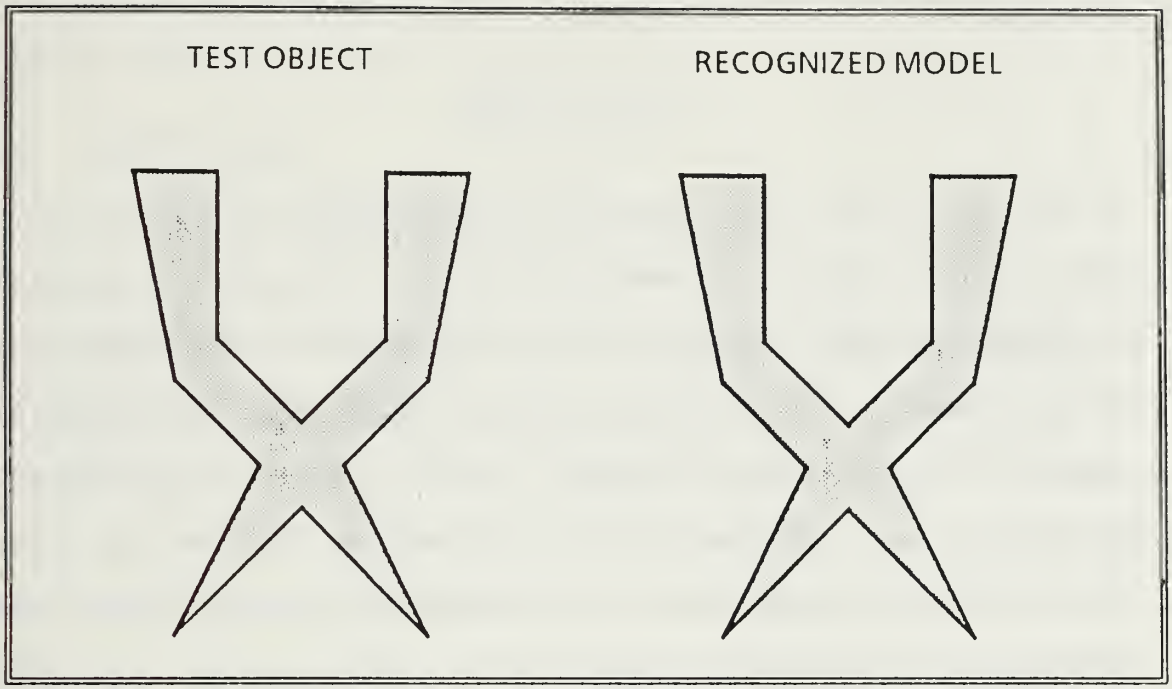


Figure 15. More Complicate Test Object and Recognized Model.

where ${}_mC_3$ is the number of different combinations of the Affine basis, and ${}_3P_3$ is the number of permutations of three arbitrary non-collinear points. When the number of interesting points of the models is small, the matching algorithm will be much faster. The computation for Hash table setup in the encoding procedure is usually done off line. If there are k model points in a test scene of m points, then the probability of not choosing a model triplet in t trials is approximately

$$P = [1 - (\frac{k}{m})^3]^t \tag{28}$$

where 3 relates to three non-collinear points. Hence, for a given $\epsilon > 0$, if we let the probability of missing the correct triplet P be less than ϵ, the number of the trials t is

$$log\,\epsilon \geq log P = log[1 - (\frac{k}{m})^3]^t \tag{29}$$

that is,

$$t \leq \frac{log\ \epsilon}{log(1-(\frac{k}{m})^3)} \tag{30}$$

In the algorithm, the procedure of recognition repeats itself for each of the selected triplets. The above inequality describes the worst case number of trials before the matching is successful. If one wants the probability of missing the correct triplet to be small, one has to do more triplet selection. Finding the interesting points is also computationally expensive. For example, the outline of a test object is obtained by chain-code in this system. The big curvature change in the linked chain is selected as an interesting point. This choice is pattern dependent. A polyhedral object in the scene naturally suggests the use of polyhedral vertices, deep concavities, and zero curvature points. Sometimes, interesting points do not have to appear physically in the image. For example, an interesting point may be taken as the intersection of two nonparallel lines that do not touch in the test object. Reducing the computation complexity can be achieved by representing a line by only one pair of interesting points if there are some other interesting points on this line. This can help by reducing the order of magnitude n in the computation. In general, Affine invariant matching is a highly parallel operation. The input test

object can be compared against all the models in the Hash table simultaneously.

C. LIMITATIONS

1. Model Representation

The major limitation of a model based vision system is the low dimensionality in spatial representation or description of models or test objects. This restricts the system's functionality to a limited class of objects observed from a few fixed viewpoints. The system developed here cannot be applied to patterns with both convex and non-convex curves. However, the algorithm still can be performed in these cases by using the *footprint* method suggested by R. Hummel and H. Wolfson [Ref. 1]. The following description is quoted from that paper.

> The *footprint* is a numerical Affine invariant *shape characteristic* that is a representation of the concavity shape. To compute the footprint, we first normalize a concavity by applying the transformation which maps its triplet basis to a standard equilateral triangle. That is, the concavity endpoints are mapped to (-1,0)(1,0), and the third point to $(0,3^{0.5})$. To each such normalized shape we assign a vector of numbers that we call the 'footprint.' One of the footprint schemes that we can use is illustrated in Figure 16.
>
> For some constant s (say $5 \leq s \leq 10$), we divide the upper half plane by $s + 1$ rays based at the origin, with angle π/s between two consecutive rays i and $i + 1$. The footprint will be s-vector $(a_1,a_2,\ldots,a_3)$, where each component is quantized into one of a number of discrete bins.
>
> We now proceed as before to construct a hash table. Each footprint is used as an entry to the hash table, where the *model* and *concavity* numbers are recorded. In the recognition phase, each concavity is used to compute a footprint, and the appropriate entry in the hash table is

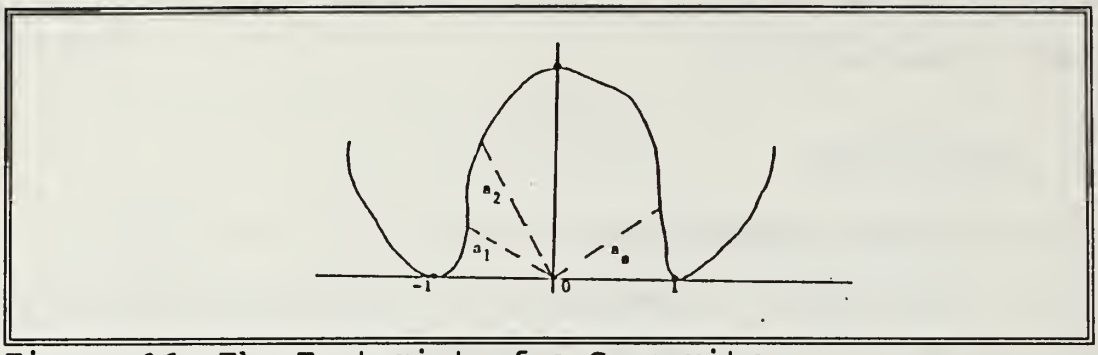


Figure 16. The Footprint of a Concavity.

accessed. For each pair, (*model*, *concavity*), appearing in the hash table as the location, we compute the appropriate Affine transformation to the associated model, and attempt to verify an instance of the model in the image.

Basically the same method for handling polygon shapes can be extended to concave shapes.

2. Quantization Problem in Generation of the Hash Key

To convert the Affine coefficient pair to *inkey*, we multiply each of the Affine coefficient pair (ξ, η) by a large constant and sum them up. This conversion causes most of the *inkeys* to be equal. Therefore, the generated *hashkeys* are not distributed uniformly. These collisions result in a long *bucket list*. Such a long bucket list is the main disadvantage of the present implementation since it slows down the hash table search speed.

3. Noise Handling

The searching in this implementation is sensitive to the noise. With noisy interesting points the test triplet will lead to the situation where the Affine coefficient pair does not hit the Hash table due to the noise. This is one of the weaknesses of this implementation. However further work

can be done to improve the solution as suggested in Wolfson's paper. [Ref. 1] The following paragraph is adopted from that paper.

> An Affine transformation in the plane is uniquely defined by the transformation of three non-collinear points. However, in practical applications this transformation may be somewhat distorted, because of noisy computation of these three points. Knowledge of additional points, which were transformed to each other may help to improve the accuracy of the computed transformation. The incorporating of the *Mean Square Error* match seems to be one of the solutions. Specifically, assume that we are looking for an Affine match between the sequences of the planar points u_i and v_i where $i = 1,2,...,n$. By using the MSE method, we could find the optimal Affine match T which will minimize the square of the distance between the sequence Tu_i and v_i where $i = 1,2,...,n$.

D. IMPRESSIONS OF THE ALGORITHM FEATURE

1. Two Phase Algorithm

A key feature of this algorithm is the division of *Modeling* (*Encoding*) and *Recognition* (*Matching*) procedures. The modeling is performed off-line. This enables the speed of an on-line recognition procedure to be optimized.

2. Hash Implementation

The method described here differs from other existing model-based matching systems. This method uses hashing technique to search for the Affine coefficient. The speed of on-line recognition depends on the size of the interesting points of the test object. The amount of on-line processing does not depend too much on the total number of models

existing in the Hash table. In other words, the recognition time is independent of the number of different models in the data-base. This advantage makes the method uniquely attractive as compared to others.

V. CONCLUSION

A. SUMMARY

This thesis begins with a survey of various model-based vision systems. Different feature extraction methods and matching techniques are also discussed. To achieve not only the ability to recognize partial occluded test objects but also to have high processing speed, the Affine invariant transformation technique is preferred. The Affine invariant matching was studied by R. Hummel and H. Wolfson. [Ref. 1] The Affine invariant matching was proposed to provide a high speed and the effective matching technique.

Motivated by this voting match technique, the main objective of this thesis was to develop a local system that can accept an incoming scene and decide where known models exist in it. Affine invariant matching handles the modeling and matching procedures independently. The use of a Hash table indexed by possible Affine coefficient pairs allows for fast content dependent matching. The program developed here is in principle similar to that in Reference 1. Some unique features of this algorithm can be listed as follows.

- Encoding and Recognition procedures are separated.
- Matching can detect objects under Affine transformation.

- Partial matching helps to detect occluded objects.
- The Hashing technique allows us to do content dependent matching efficiently.
- Many operations in the algorithm can be performed simultaneously with parallel processing.

B. EXPERIENCE GAINED

The experience gained in this study can be summarized as follows.

1. Affine Invariant

This is the characteristic of Affine invariant matching. In experiments, the algorithm can detect objects under similarity transformations. The models existing in the scene, can experience translation, rotation, or scaling to some arbitrary degree. The algorithm is able to recognize the existing models.

2. Partial Matching

The triplets used in this algorithm are essential local features. In a composite test scene objects can be recognized in the presence of occlusion due to the hidden interesting points without any problem.

3. Speed and Complexity

The computational complexity of this system is highly dependent on the number of interesting points used after the preprocessing phase. The complexity of this algorithm grows as m^4, where m is the number of interesting points. However,

if effective classification of composite local features exists, then it can be incorporated in the Encoding procedure to improve the system performance.

4. Noise Perturbation

The established system is very rigid in calculating the hashkey in the Recognition procedure. If part of the interesting points in the scene are affected by noise, the calculated Affine coefficient pairs based on those affected points will not contribute to the evidence accumulation of the model triplet. This means that the present implementation is still noise sensitive. However if the number of the unaffected interesting points is in the majority, contribution to the voting from other unaffected points still allows recognition of the test object.

C. RECOMMENDATIONS

The main weakness in the present implementation exists in the data structure and the decision-making technique.

- The data structure used in the system is not efficiently utilized due to the redundancy of the record fields of the data file, such as the two dimensional vote array. This is a physical size problem. If there are more than 14 interesting points in the scene, the system implemented here cannot handle them. If more memory is available recompilation of the source code will reduce the limitation.

- On the VERIFY module, the decision-making technique used is not yet mature. The present method just considers the corresponding edge distance between the model and the recognized model. If there are scaled or enlarged models in the test scene, the threshold used in decision-making needs to be changed from one scaled or enlarged model to

another. This needs improvement to obtain a flexible decision-making strategy.

Beside these two problems, possible subjects for future study related to Affine invariant matching could be:

- Find a more effective local feature classification technique in model representation. This can help to reduce the Hash table size and reduce the computation complexity.
- Use the mean square error method to achieve better noise immunity for the Hash table access in the recognition phase.
- Extend Affine invariant matching to handle 3-D object recognition.
- Find some parallel hardware implementation such as Content Addressable Memory (CAM) or others to achieve parallel searching or matching of a test object against various model points.

APPENDIX A

PASCAL-LIKE PSEUDO CODES

This appendix contains those Pascal-like pseudo codes used in the established system. Explanation of each of the modules is discussed in Chapter III.

```
**********************************************************************


program affine(interestingdata,hashkeydata)

(* affine finds the affine coefficient pairs *)
(* the input is interesting data file
   the output is hashkey data file *)

procedure matrix_mult(inverse_matrix : array;
                      var difference,coord :array);
begin
   for i := 1 to 2 do
   for j := 1 to 2 do
   coord[i,j] := coord[i,j] + inverse_matrix[i,n]
                 *  difference[n,j];
end;

procedure inverse(base_matrix : array;
                  var inverse_matrix :array);
begin
   for i := 1 to 2 do
   for j := 1 to 2 do
   determinent := det (base_matrix[i,j]);
   inverse_matrix[i,j] := (-1)**(i+j) * base_matrix[i,j]
                          / determinent;
end;

procedure calculate_coord(basex,basey : array);
begin
   for i := 1 to 2 do
   begin
    base_matrix[i,j] := basex[i] - basex[i+1];
    base_matrix[i,j] := basey[i] - basey[i+1];
   end;
   inverse(base_matrix,inverse_matrix);
   difference[i,j] := restbase[i] - base[3];
   matrix_mult(inverse_matrix,difference,coord);
end;

procedure perm(basex,basey :array;
               k : integer);
```

```
begin
  if k = 3 then calculate_coord(basex,basey);
  else
  for i := k to 3 do
  begin
    tx := basex[i];
    ty := basey[i];
    basex[i] := basex[k];
    basey[i] := basey[k];
    basex[k] := tx;
    basex[k] := ty;
    perm(basex,basey,k+1);
  end;
end;

procedure collinear(combinationx,combinationy : array);
begin
   t1 := arctan((combinationx[3]-combinationx[2]) /
         (combinationx[2]-combinationx[1]))
   t2 := arctan((combinationy[3]-combinationy[2]) /
         (combinationy[2]-combinationy[1]))
   if t1 = t2 then collinear := true;
   else perm(basex,basey,1)
end;

procedure combination(k,index : integer);
begin
   if index > 3 then collinear(comx,comy)
   else
   for i := k to interestingno do
      begin
         comx[index] := x[i];
         comy[index] := y[i];
         combination(i+1,index+1);
      end;
end;

(* main affine *)
begin
  while not eof(interestingdata) do
  begin
  readln(interestingdata,x[i],y[i]);
  combination(1,1);
  end;
end;

*****************************************************************


program hash(hashkeydata,hashdata);

(* hash generates the hash table *)
```

```
(* input is hashkey data file
   output is hash data file *)

procedure initialize;
var i : integer;
begin
        for i := 1 to tablesize do
        model[i].hashkey := empty;
        model[i].link := empty;
end;

procedure hashing(var hashkey : integer; inkey : integer);
const   multipler = 100000;
begin
  inkey := (key1 + key2) * multiplier;
  hashkey := (inkey mod tablesize);
  if model.hashkey <> empty then
         insert(hashkey)
  else
         collision(hashkey);
end;

procedure collision(hashkey : integer);
var bucket_length : interger;
begin
  bucket_length := 0;
  while model[hashkey].link <> empty do
  begin
    hashkey := model[hashkey].link;
    bucket_length := bucket_length + 1;
  end;
  insert(model[hashkey]);
  model[hashkey].link := bucket_length;
end;

(* main hash)
begin
  while not eof(hashkeydata) do
  begin
  readln(hashkeydata,model.inkey,model.key1,model.key2):
  initialize;
  hashing(1,model.inkey);
end.

*****************************************************************


program tally(interestingdata,cadidatedata,hashdata,plotingdata);


(* tally generates the recognized model interesting data point
        in the test object space *)
```

```
(* the input data files are
       interetingdata : the original information of the model,
       candidatedata : the test triplet, and
       hashdata : the Hash table

   the output data file is
       plotingdata *)

procedure search(modelno : integer);
begin
while (model.modelid = modelno) and (modelno <= total_modelno) do

   begin
     for i:= 1 to test_interestingno do
     begin
       cand[i].hashkey := cand[i].inkey MOD tablesize;
       if (cand[i].hashkey = model.hashkey) then
       begin
        vote(modelid,tripletid) := vote(modelid,tripletid) + 1;
        temporary := model;
        (* Temporary is used to collect the accessed entry*)
        temporary_table_size := temporary_table_size +1;
       end;
      end;
    end;
verify(1);
end;

procedure verify(tripletid);
begin
   for i := 1 to temporary_table_size do
   begin
     if (screen_threshold < vote(modelid,tripletid) then
        begin
         selected_tripletid := tripletid;
         number_selected := number_selected +1;
        end
     else
        begin
        successful := false;
        end;

(* control goes back to main program for the 2nd test triplet *)


     if (selected.modelid = temporary.modelid) and
        (selected.tripletid = temporary.tripletid) then
     begin
       collected.key1 := temporary[tripletid].key1;
       collected.key2 := temporary[tripletid].key2;
       reconstruct(number_selected,collected.key1,collected.key2,
```

```
                    selected_model_triplet,test_triplet);
      end;
    end;
 end;
search (modelno+1);
end;

procedure reconstruct(number_selected : integer;
                      key1,key2 : real;
                     selected_model_triplet,test_triplet : array);

begin

  (*   perform   the   Affine   transformation   on   the
selected_model_triplet
     and the test_triplet respectively to obtain the model
interesting
     points and the interesting points exit in the test object*)


   for k := 1 to number_selected do
   begin
   for i := 1 to interestingpoint do
   (x[i],y[i]) := affine (key1,key2,selected_model_triplet);
   for j := 1 to interestingpoint do
   (x[j],y[j]) := affine (key1,key2,test_triplet);

   (* find the order corresponding relationship between (x[i],y[i])

      and (x[j],y[j]) to obtain ordered(x[i],y[i]) *)

   (* perform the square error between the ordered(x[i],y[i])
      and the test object interesting coordinate *)

     if (model_triplet[k].square_error < square_error_threshold)
and
        (model_triplet[k].square_error  <=  minimun_square_error)
then
         begin
         optimum_triplet := k;
         minimun_square_error := model_triplet[k].square_error;
         end
     else
         begin
         verify(tripletid);
         end;
  display(optimum_model_triplet);

(* the control goes back to the search for the rest of models *)
```

```
end;

(* main tally *)
begin
  readln (interestingdata,total_modelno,interestingno);
  while not eof(hashdata) do
  readln (hashdata,model.hashkey,model.inkey,model.key1,model.key2,

         model.modelid,model.tripletid,model_triplet);
  while not eof(candidatedata) do
  begin
  readln
(candidatedata,cand.inkey,cand.key1,cand.key2,test_triplet);
  search(1);
  end;
end.
```

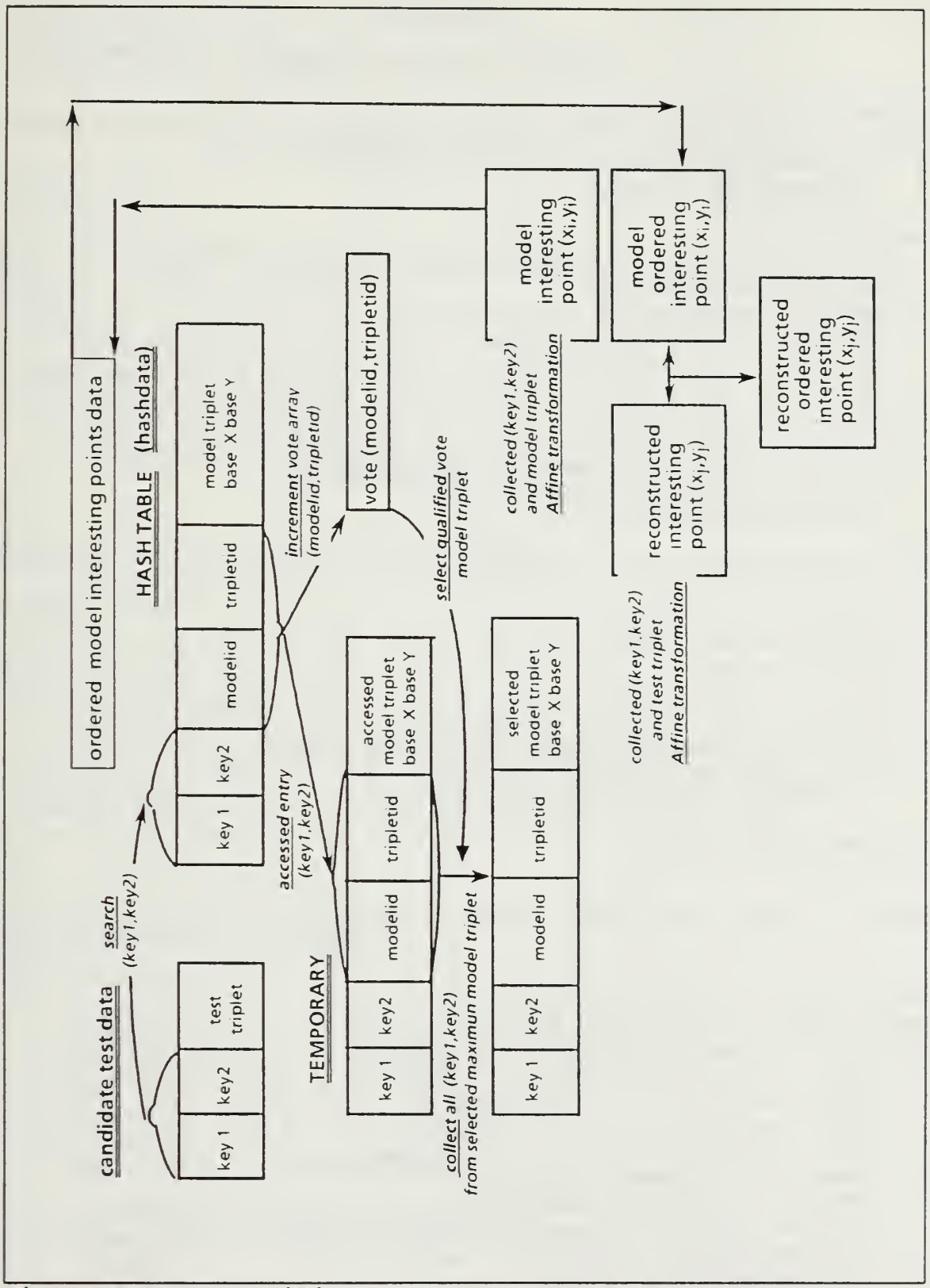

ordered model interesting points data
HASH TABLE (hashdata)
model triplet base X base Y
tripletid
modelid
key2
key 1
increment vote array (modelid,tripletid)
vote (modelid,tripletid)
model interesting point (xi,yi)
model ordered interesting point (xi,yi)
reconstructed ordered interesting point (xj,yj)
collected (key1,key2) and model triplet Affine transformation
reconstructed interesting point (xj,yj)
collected (key1,key2) and test triplet Affine transformation
candidate test data
search (key1,key2)
test triplet
accessed entry (key1,key2)
TEMPORARY
accessed model triplet base X base Y
select qualified vote model triplet
collect all (key1,key2) from selected maximun model triplet
selected model triplet base X base Y

Figure A-1. Recognition Procedure Data Flowchart.

APPENDIX B

LISTING OF SOURCE CODES

```
program gen(interesting);
type ary = array [1..2,1..50] of integer;
var i,j,modelno,interestingno : integer;
    interesting : text;
    x,y : ary;
begin
  open(interesting,'interesting.dat',history := new);
  rewrite(interesting);
{**************model 1 ************}
   modelno := 1; interestingno := 7;
   x[1,1] := 20; x[1,2] := 30; x[1,3] := 40; x[1,4] := 50;
   x[1,5] := 40; x[1,6] := 30; x[1,7] := 30;
   y[1,1] := 30; y[1,2] := 40; y[1,3] := 30; y[1,4] := 20;
   y[1,5] := 10; y[1,6] := 10; y[1,7] := 20;
   writeln(interesting,modelno,interestingno);
   for i := 1 to interestingno do
   writeln(interesting,x[1,i],y[1,i]);

{**************model 2 ************}

   modelno := 2; interestingno := 5;
   x[2,1] := 40; x[2,2] := 40; x[2,3] := 35; x[2,4] := 30;
   x[2,5] := 30;
   y[2,1] := 35; y[2,2] := 20; y[2,3] := 20; y[2,4] := 15;
   y[2,5] := 35;
   writeln(interesting,modelno,interestingno);
   for i := 1 to interestingno do
   writeln(interesting,x[2,i],y[2,i]);

  close(interesting);
end.

*****************************************************************


program preproc(interesting,coordinate);
const total = 9999;{user supply}
type ary = array [1..total] of integer;
    ary1 = array [1..2,1..2] of real;
    ary2 = array [1..3] of integer;
    ary3 = array [1..2,1..1] of real;
    ary4 = array [1..2,1..2] of integer;
var count,i,j,k,index : integer;
    outmodelno,outinterestingno : array [1..10] of integer;
    modelno,tripletno,interestingno,table_size : integer;
    x,y,rest_basex,rest_basey : ary;
    inverse_matrix : ary1;
    basex,basey,comx,comy : ary2;
```

```
    diff,coord : ary3;
    base_matrix : ary4;
    interesting,coordinate : text;

procedure matrix_mult(inverse_matrix : ary1;
                var diff,coord : ary3);
const k = 2;
      l = 2;
        m = 1;
var i,i1,n : integer;
begin
   for i := 1 to l do
     for i1 := 1 to m do
       coord[i,i1] := 0;
   for i := 1 to l do
     for i1 := 1 to m do
       for n := 1 to l do
       coord[i,i1] := coord[i,i1] + inverse_matrix[i,n] *
                      diff[n,i1];
end;

procedure inverse(base_matrix : ary4;
           var inverse_matrix : ary1);
var det : real;

begin
for i := 1 to 2 do
   det := (base_matrix[1,1] * base_matrix[2,2]) -
       (base_matrix[1,2] * base_matrix[2,1]);
   inverse_matrix[1,1] := base_matrix[2,2]/det;
   inverse_matrix[1,2] := base_matrix[1,2] * (-1) /det;
   inverse_matrix[2,1] := base_matrix[2,1] * (-1) /det;
   inverse_matrix[2,2] := base_matrix[1,1] /det;
end;

procedure calculate_coord(basex,basey : ary2);
var i,m,i1,i2,i3 : integer;
begin
   for i := 1 to 2 do
      begin
     base_matrix[1,i] := basex[i] - basex[i+1];
     base_matrix[2,i] := basey[i] - basey[i+1];
      end;
   inverse(base_matrix,inverse_matrix);
    i2 := 1;
      for i := 1 to interestingno do

{i is the number of basis triplets}

        begin
        if ((basex[1] <> x[i]) or (basey[1] <> y[i])) and
           ((basex[2] <> x[i]) or (basey[2] <> y[i])) and
```

```
          ((basex[3] <> x[i]) or (basey[3] <> y[i])) then
         begin
           rest_basex[i2] := x[i];
           rest_basey[i2] := y[i];
           i2 := i2 + 1;
         end;
        end;
   for i := 1 to (interestingno-3) do
     begin
       diff[1,1] := rest_basex[i] - basex[3];
       diff[2,1] := rest_basey[i] - basey[3];
     matrix_mult(inverse_matrix,diff,coord);
     count := count +1;
        i3 := count mod (interestingno-3);
     if i3 = 1 then tripletno := tripletno + 1;
       write(coordinate,modelno,tripletno,coord[1,1],coord[2,1]);

     for i1 := 1 to 3 do
     write(coordinate,basex[i1],basey[i1]);
     writeln(coordinate);
     end;
end;

procedure perm(basex,basey : ary2;
           k : integer);
var i,tx,ty : integer;
begin
   if k = 3 then
      calculate_coord(basex,basey)
   else
      for i := k to 3 do
        begin
         tx := basex[i];
      ty := basey[i];
      basex[i] := basex[k];
      basey[i] := basey[k];
      basex[k] := tx;
      basey[k] := ty;
      perm(basex,basey,k+1);
       end;
end;

procedure colinear(comx,comy : ary2);
var colinearx1,colinearx2,colineary1,colineary2 : integer;
    t1,t2 : real;
begin
      colinearx1 := comx[2] - comx[1];
      colinearx2 := comx[3] - comx[2];
      colineary1 := comy[2] - comy[1];
      colineary2 := comy[3] - comy[2];
       if (colinearx1 <> 0) then
      t1 := arctan(colineary1 / colinearx1)
```

```
       else t1 := 1.5707;
       if (colinearx2 <> 0) then
      t2 := arctan(colineary2 / colinearx2)
       else t2 := 1.5707;
   if t1 <> t2 then perm(comx,comy,1);
end;

procedure comb(k,index : integer);{use the recursive}
var i : integer;
begin
   if index > 3 then colinear(comx,comy)

   else
      for i := k to interestingno do
     begin
        comx[index] := x[i];
        comy[index] := y[i];
        comb(i+1,index+1);
     end;
end;

begin{main}
   open(interesting,'interesting.dat',history := old);
   reset(interesting);
   open(coordinate,'coordinate.dat',history := new);
   rewrite(coordinate);
   table_size := 0;j:= 1;
   while not eof(interesting) do
     begin
       readln(interesting,outmodelno[j],outinterestingno[j]);
       modelno := outmodelno[j];
       interestingno := outinterestingno[j];
       for i := 1 to interestingno do
        readln(interesting,x[i],y[i]);
       count :=0;tripletno := 0;j := j+1;
       comb(1,1);
       table_size := table_size + count;
     end;
 writeln(coordinate,table_size,outmodelno[1],outinterestingno[1],

     outmodelno[2],outinterestingno[2]);
   close(interesting);
   close(coordinate);
end.

*****************************************************************


program keycon(infile,outfile);
type
    inrec = record
     modelno,tripletno : integer;
```

```
     key1,key2 : real;
     basex1,basey1,basex2,basey2,basex3,basey3 : integer
    end;
    outrec = record
     outkey : integer;
     basex1,basey1,basex2,basey2,basex3,basey3 : integer
    end;
    ary1 = array [0..99999] of integer;
var
    a : array[0..99999] of outrec;
    b : inrec;
    infile,outfile : text;
    temp : real;
    i,j,j1 : integer;
    outmodelno,outinterestingno : array [1..50] of integer;
    table_size,ary : ary1;

procedure prime(var primeno : integer);
var
    is : boolean;
    i,j,k,c : integer;
begin
   ary[1] := 2;
   c:= 1;
   for i := 3 to primeno do
   begin
     j := 1;
     is := false;
     repeat
      if (i mod ary[j]) =0 then
     is := true;
      j:= j+1;
     until (ary[j] = 0) or (ary[j] > sqrt(i)) or is;
     if not is then
      begin
     c := c+1;
     ary[c] :=i;
      end;
    end;
  is:=false;
  while not is do
  begin
   for k := 1 to j do
   if(primeno mod ary[k]) = 0 then
   is := true;
   primeno := primeno+1;
  end;
end;

begin
open (infile,'coordinate.dat',history:=old);
reset(infile);
```

```
open (outfile,'hash_key.dat',history:=new);
rewrite(outfile);
   j :=0;
   while not eof(infile) do
     begin
        readln(infile,table_size[j]);
         j := j+1;

     end;
close(infile);

open (infile,'coordinate.dat',history:=old);
reset(infile);
   j1 :=1;
   while not eof(infile) do
    begin

    if j1 = table_size[j-1]+1 then
     begin
 readln(infile,table_size[j-1],outmodelno[1],outinterestingno[1],

          outmodelno[2],outinterestingno[2]);
     end
    else
    readln(infile);
    j1 := j1+1;
    end;
     prime(table_size[j-1]);
     write(outfile,table_size[j-1]);
     write(outfile,outmodelno[1],outinterestingno[1],
     outmodelno[2],outinterestingno[2]);
     writeln(outfile);
close(infile);

open (infile,'coordinate.dat',history:=old);
reset(infile);
        i := 0;
     while i <> j-1 do
          with b do
          begin
              readln(infile,modelno,tripletno,key1,key2,
                    basex1,basey1,basex2,basey2,basex3,basey3);
              temp := (key1+key2)*10000000;
              a[i].outkey := trunc (temp);
              writeln(outfile,a[i].outkey,modelno,
                    tripletno,key1,key2,basex1,basey1,basex2,
                    basey2,basex3,basey3);
              i := i+1
          end;
close(infile);
close(outfile);
end.
```

```
**********************************************************************

program hashing(input,output);
const database_modelno = 2;
type
     rec = record
     inkey,modelno,tripletno,link : integer;
     key1,key2 : real;
     basex1,basey1,basex2,basey2,basex3,basey3 : integer;
     end;
   outrec = record
     hashkey,inkey,modelno,tripletno,link : integer;
     key1,key2 : real;
     basex1,basey1,basex2,basey2,basex3,basey3 : integer;
     end;
   var  a : array[0..99999] of outrec;
     b: rec;
     i,j,k,table_size : integer;
     outmodelno,outinterestingno : array [1..database_modelno]
                                        of integer;

   procedure print;
   var i1 : integer;
   begin
     for i1 := 0 to table_size do
       begin
         if a[i1].hashkey <> -1 then
          with a[i1] do
            writeln(output,hashkey,inkey,modelno:2,
              tripletno:4,key1,key2,basex1:4,
               basey1:4,basex2:4,basey2:4,basex3:4,basey3:4,link)

       end
   end;

   procedure solve_collision(var i,j : integer);

{if the hash key is same then find the last same hash key of the

linked chain thru the link}

   begin
     while a[i].link <> -1 do
       begin
         i := a[i].link
     end;

{test the last hash key of the linked chain if its next position

points to the free entry, till find the free space
```

```
i : means the previous linked same hash key
j : means the one which is free entry, it also link to previous

    same key}
     j := i;
     while a[j].hashkey <> -1 do
       begin
         j := j+1;
         if j > table_size then j := 0
     end;{while}
   end;

   procedure initialize;
   var i1,i2 : integer;
   begin
     for i1 := 0 to table_size do
     with a[i1] do
        a[i1].hashkey :=-1;
     for i2 := 0 to table_size do
     with a[i2] do
        a[i2].link :=-1;
  end;

   procedure insert(var i,j:integer);
   begin
     if (a[i].hashkey = -1)  then
        begin
          a[i].hashkey :=i;
          a[i].inkey := b.inkey;
          a[i].modelno := b.modelno;
          a[i].tripletno := b.tripletno;
          a[i].key1 := b.key1;
          a[i].key2 := b.key2;
          a[i].basex1 := b.basex1;
          a[i].basey1 := b.basey1;
          a[i].basex2 := b.basex2;
          a[i].basey2 := b.basey2;
          a[i].basex3 := b.basex3;
          a[i].basey3 := b.basey3;
        end
     else
        begin
          solve_collision(i,j);
          if (a[i].modelno <> a[j].modelno) and
             (a[i].tripletno <> a[j].tripletno) then
               begin
                    a[i].link := j;
                    a[j].hashkey := j;
                    a[j].inkey := b.inkey;
                    a[j].modelno := b.modelno;
                    a[j].tripletno := b.tripletno;
                    a[j].key1 := b.key1;
```

```
                    a[j].key2 := b.key2;
                    a[j].basex1 := b.basex1;
                    a[j].basey1 := b.basey1;
                    a[j].basex2 := b.basex2;
                    a[j].basey2 := b.basey2;
                    a[j].basex3 := b.basex3;
                    a[j].basey3 := b.basey3;
                end
        end
    end;

    begin (* main *)
      open (input,'hash_key.dat',history := old);
      reset(input);
      open (output,'hash.dat',history := new);
      rewrite(output);
      readln(input,table_size,outmodelno[1],outinterestingno[1]
           ,outmodelno[2],outinterestingno[2]);
      writeln(output,table_size,outmodelno[1],outinterestingno
                 [1],outmodelno[2],outinterestingno[2]);
        initialize;
      i :=0;j := 0;
      while not eof(input) do
        begin
         with b do
           begin
           read(input,inkey);
           read(input,modelno,tripletno,key1,key2,
           basex1,basey1,basex2,basey2,basex3,basey3);
           readln;
           i := b.inkey mod table_size;
           insert(i,j);
           end
        end (*while*);
      print;
      close(input);
      close(output)
    end.

**********************************************************************


program test_gen(interesting);

type ary = array [1..2,1..50] of integer;
var i,j,modelno,interestingno : integer;
     interesting : text;
     x,y : ary;
begin
  open(interesting,'tally_interesting.dat',history := new);
  rewrite(interesting);
{*********************test object**********************}
```

```
   modelno:=1;interestingno := 10;
   x[1,1] := 50; x[1,2] := 40; x[1,3] := 30;
   x[1,4] := 10; x[1,5] := 10;
   x[1,6] := 20; x[1,7] := 30; x[1,8] := 30; x[1,9] := 40;
   x[1,10] := 40;
   y[1,1] := 10; y[1,2] := 0; y[1,3] := 10;
   y[1,4] := 10; y[1,5] := 40;
   y[1,6] := 40; y[1,7] := 50; y[1,8] := 30; y[1,9] := 30;
   y[1,10] := 20;
   writeln(interesting,modelno,interestingno);
   for i := 1 to interestingno do
   writeln(interesting,x[1,i],y[1,i]);

{*******************alternative data set**********************}
{   y[1,1] := 90; y[1,2] := 70; y[1,3] := 50; y[1,4] := 30;
    y[1,5] := 40; y[1,6] := 30; y[1,7] := 50; y[1,8] := 70;
    y[1,9] := 90; y[1,10] := 90; y[1,11] := 70; y[1,12] := 50;
    y[1,13] := 70; y[1,14] := 90;
   writeln(interesting,modelno,interestingno);
   for i := 1 to 14 do
   writeln(interesting,x[1,i],y[1,i]);
}
   close(interesting);
end.

*****************************************************************


program test_preproc(interesting,coordinate);
const total = 9999;{user supply}
type ary = array [1..total] of integer;
    ary1 = array [1..2,1..2] of real;
    ary2 = array [1..3] of integer;
    ary3 = array [1..2,1..1] of real;
    ary4 = array [1..2,1..2] of integer;
var count,i,k,index : integer;
    modelno,interestingno,table_size : integer;
    x,y,rest_basex,rest_basey : ary;
    inverse_matrix : ary1;
    basex,basey,comx,comy : ary2;
    diff,coord : ary3;
    base_matrix : ary4;
    interesting,coordinate : text;

procedure test_matrix_mult(inverse_matrix : ary1;
                var diff,coord : ary3);
const k = 2;
      l = 2;
        m = 1;
var i,j,n : integer;
begin
   for i := 1 to l do
```

```
      for j := 1 to m do
        coord[i,j] := 0;
    for i := 1 to l do
      for j := 1 to m do
        for n := 1 to l do
        coord[i,j] := coord[i,j] + inverse_matrix[i,n] *
                      diff[n,j];
end;

procedure test_inverse(base_matrix : ary4;
            var inverse_matrix : ary1);
var det : real;

begin
for i := 1 to 2 do
   det := (base_matrix[1,1] * base_matrix[2,2]) -
       (base_matrix[1,2] * base_matrix[2,1]);
   inverse_matrix[1,1] := base_matrix[2,2]/det;
   inverse_matrix[1,2] := base_matrix[1,2] * (-1) /det;
   inverse_matrix[2,1] := base_matrix[2,1] * (-1) /det;
   inverse_matrix[2,2] := base_matrix[1,1] /det;
end;

procedure test_calculate_coord(basex,basey : ary2);
var i,j,i1,i2 : integer;
begin
   for i := 1 to 2 do
      begin
     base_matrix[1,i] := basex[i] - basex[i+1];
     base_matrix[2,i] := basey[i] - basey[i+1];
      end;
   test_inverse(base_matrix,inverse_matrix);
   i2 := 1;
   for i := 1 to interestingno do

{this i is the number of basis triplet}

     begin
        if ((basex[1] <> x[i]) or (basey[1] <> y[i])) and
         ((basex[2] <> x[i]) or (basey[2] <> y[i])) and
         ((basex[3] <> x[i]) or (basey[3] <> y[i])) then
          begin
            rest_basex[i2] := x[i];
            rest_basey[i2] := y[i];
            i2 := i2 +1;
          end;
     end;
    for i := 1 to interestingno-3 do
        begin
       diff[1,1] := rest_basex[i] - basex[3];
       diff[2,1] := rest_basey[i] - basey[3];
   test_matrix_mult(inverse_matrix,diff,coord);
```

```
      count := count +1;
      write(coordinate,modelno,count,coord[1,1],coord[2,1]);
    for i1 := 1 to 3 do
      write(coordinate,basex[i1],basey[i1]);
      writeln(coordinate);
      end;
end;

procedure test_perm(basex,basey : ary2;
           k : integer);
var i,tx,ty : integer;
begin
   if k = 3 then
      test_calculate_coord(basex,basey)
   else
      for i := k to 3 do
        begin
         tx := basex[i];
      ty := basey[i];
      basex[i] := basex[k];
      basey[i] := basey[k];
      basex[k] := tx;
      basey[k] := ty;
      test_perm(basex,basey,k+1);
       end;
end;

procedure test_colinear(comx,comy : ary2);
var colinearx1,colinearx2,colineary1,colineary2 : integer;
    t1,t2 : real;
begin
      colinearx1 := comx[2] - comx[1];
      colinearx2 := comx[3] - comx[2];
      colineary1 := comy[2] - comy[1];
      colineary2 := comy[3] - comy[2];
       if (colinearx1 <> 0) then
      t1 := arctan(colineary1 / colinearx1)
       else t1 := 1.5707;
       if (colinearx2 <> 0) then
      t2 := arctan(colineary2 / colinearx2)
       else t2 := 1.5707;
   if t1 <> t2 then test_perm(comx,comy,1);
end;

procedure comb(k,index : integer);{use the recursive}
var i : integer;
begin
   if index > 3 then test_colinear(comx,comy)

   else
      for i := k to interestingno do
     begin
```

```
        comx[index] := x[i];
        comy[index] := y[i];
        comb(i+1,index+1);
      end;
end;

begin{main}
   open(interesting,'tally_interesting.dat',history := old);
   reset(interesting);
   open(coordinate,'tally_coordinate.dat',history := new);
   rewrite(coordinate);
   table_size := 0;
   while not eof(interesting) do
     begin
       readln(interesting,modelno,interestingno);
       writeln(coordinate,interestingno);
       for i := 1 to interestingno do
        readln(interesting,x[i],y[i]);
       count :=0;
       comb(1,1);
       table_size := table_size + count;
     end;
   close(interesting);
   close(coordinate);
end.

*****************************************************************


program test_keycon(infile,outfile);
type
    inrec = record
     modelno,tripletno : integer;
     key1,key2 : real;
     basex1,basey1,basex2,basey2,basex3,basey3 : integer
    end;
    outrec = record
     outkey : integer;
     basex1,basey1,basex2,basey2,basex3,basey3 : integer
    end;
var
    a : array[0..99999] of outrec;
    b : inrec;
    infile,outfile : text;
    temp : real;
    i,j,interestingno : integer;
    table_size : array[0..99999] of integer;
begin
open (infile,'tally_coordinate.dat',history:=old);
reset(infile);
open (outfile,'tally.dat',history:=new);
rewrite(outfile);
```

```
   j :=0;
      i := 0;
     readln(infile,interestingno);
     writeln(outfile,interestingno);
     while not eof(infile) do
          with b do
          begin
              readln(infile,modelno,tripletno,key1,key2,
                    basex1,basey1,basex2,basey2,basex3,basey3);
              temp := (key1+key2)*10000000;
              a[i].outkey := trunc (temp);
              writeln(outfile,a[i].outkey,key1,key2,
              basex1,basey1,basex2,basey2,basex3,basey3);
              i := i+1
          end;
close(infile);
close(outfile);
end.

*****************************************************************


program tally(input,candfile,output);
const database_modelno = 2;
type candrec = record
     inkey : integer;
     key1,key2 : real;
     basex1,basey1,basex2,basey2,basex3,basey3 : integer;
     end;
     outrec = record
     hashkey,inkey : integer;
     key1,key2 : real;
     basex1,basey1,basex2,basey2,basex3,basey3 : integer;
     end;
     modelrec = record
     hashkey,inkey,modelno,tripletno,link : integer;
     key1,key2 : real;
     basex1,basey1,basex2,basey2,basex3,basey3 : integer;
     end;
     collected_modelrec = record
     hashkey,inkey,modelno,tripletno,link : integer;
     key1,key2 : real;
     basex1,basey1,basex2,basey2,basex3,basey3 : integer;
     vote1 : integer;
     end;
   var  initimodel1,temporary_model,temp : array[0..9999] of
                                           modelrec;
        model : array [0..23000] of modelrec;
     collected_model : array[1..9999] of collected_modelrec;
     cand: array[0..19999] of candrec;
        cand1 : array [1..9999,1..2] of candrec;
     i,j,j1,k1,key,m,index,count : integer;
```

```
        n,maximun1,maximun2 : integer;
      table_size,interestingno,test_interestingno_3 : integer;
      candfile : text;
      vote : array[0..database_modelno,0..27999] of integer;
      outmodelno,outinterestingno : array [1..database_modelno]
                                        of integer;

    procedure initialize1;
    var i,j : integer;
    begin
      for i := 0 to database_modelno do
      for j := 0 to table_size do
      vote[i,j] := 0;
   end;

    procedure initialize2;
    var i : integer;
    begin
      for i := 0 to table_size do
      with model[i] do
           hashkey := -1;
    end;

    procedure initialize3(m : integer);
    var i : integer;
    begin
      for i := 1 to m do
        with temporary_model[i] do
        temporary_model[i] := initimodel1[i];
    end;


    procedure print_model_triplet(i :integer);

(* store the information of the selected triplet *)

    begin
      with collected_model[i] do
        writeln(output,hashkey,inkey,modelno:2,tripletno:4,
           key1,key2,basex1:4,basey1:4,basex2:4,basey2:4,
           basex3:4,basey3:4,vote1);
    end;


    procedure print_test_triplet(i,k : integer);

(* store the information of the test triplet *)
(* i is the corresponding between select model triplet and test

triplet *)
    begin
         with cand1[i,k] do
```

```
     writeln(output,' ':10,inkey,' ':6,key1,key2,basex1:4,
        basey1:4,basex2:4,basey2:4,basex3:4,basey3:4);
   end;


   procedure all_maximun_selected(n : integer);

(* store all the selected model triplets even from different
model *)

   var   i,j,cout : integer;
   begin
    for i := 1 to n do
        begin
        with collected_model[i] do
          if ((modelno = 1) and (vote1 = maximun1))
              or
          ((modelno = 2) and (vote1 = maximun2))
          then
          begin
          print_model_triplet(i);
               print_test_triplet(i,1);
             end;
        end;
   end;


   procedure verify(j,j1,m : integer;var n : integer);

(* collect all the accessed model *)
(* n is index value of all selected model *)

   var i,k,k1,l : integer;
   begin
     k1 := 1;
     for i := 1 to m do
     with temporary_model[i] do
     if (modelno = temporary_model[j1].modelno) and
        (tripletno = temporary_model[j1].tripletno) then
        begin
      collected_model[n].hashkey := temporary_model[i].hashkey;
      collected_model[n].inkey := temporary_model[i].inkey;
      collected_model[n].modelno := temporary_model[i].modelno;
      collected_model[n].tripletno := temporary_model[i].
                                      tripletno;
      collected_model[n].key1 := temporary_model[i].key1;
      collected_model[n].key2 := temporary_model[i].key2;
      collected_model[n].basex1 := temporary_model[i].basex1;
      collected_model[n].basey1 := temporary_model[i].basey1;
      collected_model[n].basex2 := temporary_model[i].basex2;
      collected_model[n].basey2 := temporary_model[i].basey2;
      collected_model[n].basex3 := temporary_model[i].basex3;
```

```
      collected_model[n].basey3 := temporary_model[i].basey3;
      collected_model[n].vote1 :=
      vote[temporary_model[i].modelno,temporary_model[i].
      tripletno];
      for k := (j - test_interestingno_3) to j-1 do
            if (cand[k].inkey = temporary_model[i].inkey) and
            (cand[k].key1 = temporary_model[i].key1) and
            (cand[k].key2 = temporary_model[i].key2) then
          begin
          cand1[n,1] := cand[k];

          k1 := k1 + 1;
          end;
        n := n+1;
    end;
    initialize3(m);
  end;


  procedure find_maximun(m,j : integer;var j1 : integer);

(* j1 is the selected model triplet index value *)
  var i : integer;
      maximun : integer;
  begin
      maximun :=0;
      for i := 1 to m do
    begin
      if (vote[temporary_model[i].modelno,temporary_model[i].
         tripletno] > maximun) then
      begin
        j1 :=i;
        maximun :=
        vote[temporary_model[i].modelno,temporary_model[i].
        tripletno];
(* here the 1 and 2 as the all different models in database *)
(* each model has it's own maximun vote *)

          if (temporary_model[i].modelno = 1) and
             (maximun > maximun1)
          then maximun1 := maximun;
          if (temporary_model[i].modelno = 2) and
             (maximun > maximun2)
          then maximun2 := maximun;
      end;
      end;
  end;


  procedure search(j,key:integer;var m : integer);

(* m is the length of kth hash key bucket *)
```

```
   var i,i2 : integer;
   begin
     i := key;
     while (i <> -1)  do
        begin
          if (cand[j].inkey = model[i].inkey) and
             (cand[j].key1 = model[i].key1) and
             (cand[j].key2 = model[i].key2) then
                begin
               temporary_model[m] := model[i];
               with temporary_model[m] do
               vote[modelno,tripletno] := vote[modelno,
                                                  tripletno]+1;

               m := m +1;
                end;
          i := model[i].link;
        end;
   end;

   begin (* main *)
     open (input,'hash.dat',history := old);
     reset(input);
     open (candfile,'tally.dat',history := old);
     reset(candfile);
     open (output,'vote.dat',history := new);
     rewrite(output);
        readln(input,table_size,outmodelno[1],outinterestingno[1]

          ,outmodelno[2],outinterestingno[2]);
        readln(candfile,interestingno);
        test_interestingno_3 := interestingno - 3;
       initialize1;initialize2;
     i :=0;j := 0;
     while (i < table_size) and (not eof(input)) do
      begin
          with temp[i] do
          begin
              readln(input,hashkey,inkey,modelno,tripletno,
                 key1,key2,
           basex1,basey1,basex2,basey2,basex3,basey3,link);
          end;
          index := temp[i].hashkey;
          model[index] := temp[i];
         end; (*while*)
         m := 1;j1 :=1;n := 1;
(* ********************CANDIDATE FILE*****************)
     while not eof(candfile) do
       begin
        with cand[j] do
          begin
```

```
          readln(candfile,inkey,key1,key2,basex1,basey1,
                     basex2,basey2,basex3,basey3);
          end;
        key := cand[j].inkey mod table_size;
        search(j,key,m);
        j := j+1;
           count := j mod test_interestingno_3;
           if count = 0 then
             begin
          find_maximun(m,j,j1);
          verify(j,j1,m,n);
          m := 1;
          initialize1;
          end;
       end;
          all_maximun_selected(n);
(*********************VOTEING RESULT*******************)
     close(input);
     close(candfile);
     close(output)
   end.

*****************************************************************


program reconstruct(interesting,vote,ploting);
const   totalmodelno = 2;
type
     ary1 = array [1..10] of integer;
     ary2 = array [1..999] of integer;
     ary3 = array [1..99] of real;
     ary4 = array [1..10,1..99] of integer;
     ary5 = array [1..3] of integer;
     ary6 = array [1..2,1..2] of integer;
     ary7 = array [1..2,1..2] of real;
     ary8 = array [1..2,1..1] of real;
var
     orimodelno : ary1;
     select_basex,select_basey,test_basex,test_basey,rest_basex,

        rest_basey : ary2;
     hashkey,inkey,modelno,tripletno : integer;
     recogmodelno,recoginteresting,vot : integer;
     collected_key1,collected_key2 : real;
     test_recogx,test_recogy,ordered_test_inter_coordx,
        ordered_test_inter_coordy,
        ordered_test_basex,ordered_test_basey : ary3;
     x,y,oriinterestno : ary4;
     oritx,ority : ary5;
     base_matrix : ary6;
        inverse_matrix : ary7;
     diff,coord : ary8;
```

```
     deletemodelno :  array [0..totalmodelno] of integer;
     interesting,vote,ploting : text;
     korder,jr : integer;
     i : integer;


procedure matrix_mult(inverse_matrix : ary7;
                var diff,coord : ary8);

{this is for the matrix multiplication of inverse of
        x(1-2) x(2-3)         restx(i)-x(3)
        y(1-2) y(2-3)   and   resty(i)-y(3)}

const k = 2;
      l = 2;
        m = 1;
var i,i1,n : integer;
begin
   for i := 1 to l do
     for i1 := 1 to m do
       coord[i,i1] := 0;
   for i := 1 to l do
     for i1 := 1 to m do
       for n := 1 to l do
       coord[i,i1] := coord[i,i1] + inverse_matrix[i,n] *
                      diff[n,i1];
end;


procedure inverse(base_matrix : ary6;
            var inverse_matrix : ary7);

{find the inverse matrix of x(1-2) x(2-3)
                            y(1-2) y(2-3)}

var det : real;

begin
   det := (base_matrix[1,1] * base_matrix[2,2]) -
       (base_matrix[1,2] * base_matrix[2,1]);
   inverse_matrix[1,1] := base_matrix[2,2]/det;
   inverse_matrix[1,2] := base_matrix[1,2] * (-1) /det;
   inverse_matrix[2,1] := base_matrix[2,1] * (-1) /det;
   inverse_matrix[2,2] := base_matrix[1,1] /det;
end;



procedure calculate_coord(recogmodelno : integer;
                      cbasex,cbasey : ary5);

{calculate the /xi and /eta of the
```

```
(select_basex,select_basey) in of the model}

var i,m,i1,i2,i3 : integer;
begin
   for i := 1 to 2 do
      begin
     base_matrix[1,i] := cbasex[i] - cbasex[i+1];
     base_matrix[2,i] := cbasey[i] - cbasey[i+1];
      end;
   inverse(base_matrix,inverse_matrix);
    i2 := 1;
      for i := 1 to oriinterestno[orimodelno[recogmodelno],
               recogmodelno] do
        begin
        if ((cbasex[1] <> x[recogmodelno,i])
        or (cbasey[1] <> y[recogmodelno,i])) and
           ((cbasex[2] <> x[recogmodelno,i])
           or (cbasey[2] <> y[recogmodelno,i])) and
           ((cbasex[3] <> x[recogmodelno,i])
           or (cbasey[3] <> y[recogmodelno,i])) then
          begin
            rest_basex[i] := x[recogmodelno,i];
            rest_basey[i] := y[recogmodelno,i];
               diff[1,1] := rest_basex[i] - cbasex[3];
               diff[2,1] := rest_basey[i] - cbasey[3];
               matrix_mult(inverse_matrix,diff,coord);
               ordered_test_inter_coordx[i]:=coord[1,1]*
               (ordered_test_basex[1]-ordered_test_basex[2])+
               coord[2,1]*(ordered_test_basex[2]-
                           ordered_test_basex[3])+
               ordered_test_basex[3];
               ordered_test_inter_coordy[i]:=coord[1,1]*
               (ordered_test_basey[1]-ordered_test_basey[2])+
               coord[2,1]*(ordered_test_basey[2]-
                          ordered_test_basey[3])+
               ordered_test_basey[3];
               end;
     end;
   for i := 1 to oriinterestno[orimodelno[recogmodelno],
                               recogmodelno] do
   writeln(ploting,ordered_test_inter_coordx[i],
           ordered_test_inter_coordy[i]);
end;


procedure colinear(var comx,comy : ary5;
            var t1,t2 : real);

{using tangent y/x to test if 3 points are colinear?
t1 and t2 are the tangent value of any 2 pairs of coordinates}

var colinearx1,colinearx2,colineary1,colineary2 : integer;
```

```
begin
      colinearx1 := comx[2] - comx[1];
      colinearx2 := comx[3] - comx[2];
      colineary1 := comy[2] - comy[1];
      colineary2 := comy[3] - comy[2];
       if (colinearx1 <> 0) then
      t1 := arctan(colineary1 / colinearx1)
       else t1 := 1.5707;
       if (colinearx2 <> 0) then
      t2 := arctan(colineary2 / colinearx2)
       else t2 := 1.5707;
end;


procedure pretest(i,j : integer;
            var t1,t2 : real);

{prepareing for colinear testing}

var count : integer;
begin
  count := 1;
  while ((test_recogx[j]<>0) or (test_recogy[j]<>0)) and
        ( count < 4 ) do
     begin
     oritx[count]:=x[i,j];
     ority[count]:=y[i,j];
     ordered_test_inter_coordx[j] := test_recogx[j];
     ordered_test_inter_coordy[j] := test_recogy[j];
     ordered_test_basex[count] := test_recogx[j];
     ordered_test_basey[count] := test_recogy[j];
     j := j+1;
     count := count+1;
     end;
   colinear(oritx,ority,t1,t2);
end;


procedure mapping(bx,by : real;
           var ordering : integer;
                i : integer);

{make the first record data from the vote corresponding to
that in the second record bx,by is the input (select_basex
select_basey) or the calculated rest interesting point in
the model ordering is to pass back to match the interesting
point between model and recognized model}

var
     j : integer;
begin
    for j := 1 to oriinterestno[orimodelno[i],i] do
```

```
     begin
     if (abs(bx - x[i,j]) < 0.00001) and
           (abs(by - y[i,j]) < 0.00001) then
       begin
            ordering := j;
       end;
end;
end;


procedure interest;

{read into the original model interesting number and their
coordinates}

var
     i,j,k,k1 : integer;
begin
   open(interesting,'interesting.dat',history := old);
   reset(interesting);
   open(ploting,'ploting.dat',history := new);
   rewrite(ploting);
   i:=1;
     while not eof(interesting) do
       begin
         read(interesting,orimodelno[i]);
      readln(interesting,oriinterestno[orimodelno[i],i]);
      for j := 1 to oriinterestno[orimodelno[i],i] do
            readln(interesting,x[orimodelno[i],j],
                  y[orimodelno[i],j]);
      i := i+1;
      korder :=i;
       end;
   for k:= 1 to orimodelno[korder-1] do
   for k1 := 1 to oriinterestno[orimodelno[k],k] do
end;


procedure recognized;

{read the recognized model i.e., vote data find the order in
the recognized model}

var
     i,i1,i2,j,j1,k : integer;
     select_model_tempx,select_model_tempy,t1,t2 :real;
begin
  open (vote,'vote.dat',history := old);
  reset(vote);
  i :=1;
  while (not eof(vote)) do
  begin
```

```
    readln(vote,hashkey,inkey,modelno,tripletno,collected_key1,
           collected_key2,
           select_basex[1],select_basey[1],select_basex[2],
           select_basey[2],select_basex[3],select_basey[3],vot);

    readln(vote,inkey,collected_key1,collected_key2,
           test_basex[1],test_basey[1],test_basex[2],
           test_basey[2],test_basex[3],test_basey[3]);
    recogmodelno := modelno;
     if (recogmodelno <> deletemodelno[recogmodelno]) and
        (vot/(oriinterestno[orimodelno[recogmodelno],
         recogmodelno]-3) >= 0.5)
        then

{The vote number must be less than original model interesting
number - 3 and greater than 1 or a relative ratio ,i.e. vote
number vs. original interesting number - 3, must greater or
equal to threshold 0.5}
{        (((oriinterestno[orimodelno[recogmodelno],
         recogmodelno]-3) >= vot) and
        (vot > 1)) then}

        begin
       for J:=1 to 3 do
       begin
         mapping(select_basex[j],select_basey[j],jr,
                    recogmodelno);
         test_recogx[jr] := test_basex[j];
         test_recogy[jr] := test_basey[j];
       end;
          select_model_tempx  :=  collected_key1*(select_basex[1]-

              select_basex[2])+  collected_key2*(select_basex[2]-

                  select_basex[3])+select_basex[3];
          select_model_tempy  :=  collected_key1*(select_basey[1]-

              select_basey[2])+  collected_key2*(select_basey[2]-

                  select_basey[3])+select_basey[3];
       mapping(select_model_tempx,select_model_tempy,jr,
                   recogmodelno);
       test_recogx[jr] := collected_key1*(test_basex[1]-
                   test_basex[2])+
                   collected_key2*(test_basex[2]-test_basex[3])
             +test_basex[3];
          test_recogy[jr] := collected_key1*(test_basey[1]-
                   test_basey[2])+
                   collected_key2*(test_basey[2]-test_basey[3])
             +test_basey[3];
       for j1 := 1 to vot-1 do
         begin
```

```
          readln(vote,hashkey,inkey,modelno,tripletno,
                  collected_key1,collected_key2,
             select_basex[1],select_basey[1],select_basex[2],
             select_basey[2],select_basex[3],select_basey[3]);
             readln(vote,inkey,collected_key1,collected_key2,
             test_basex[1],test_basey[1],test_basex[2],
             test_basey[2],test_basex[3],test_basey[3]);
          select_model_tempx  :=  collected_key1*(select_basex[1]-

             select_basex[2])+  collected_key2*(select_basex[2]-

              select_basex[3])+select_basex[3];
          select_model_tempy  :=  collected_key1*(select_basey[1]-

             select_basey[2])+  collected_key2*(select_basey[2]-

              select_basey[3])+select_basey[3];

       mapping(select_model_tempx,select_model_tempy,jr,
                 recogmodelno);

          test_recogx[jr] := collected_key1*(test_basex[1]-
               test_basex[2])+collected_key2*
               (test_basex[2]-test_basex[3])+test_basex[3];
          test_recogy[jr] := collected_key1*(test_basey[1]-
               test_basey[2])+collected_key2*
               (test_basey[2]-test_basey[3])+test_basey[3];
         end;
          recoginteresting := vot+3;
       writeln(ploting,recogmodelno,oriinterestno[
                       orimodelno[recogmodelno],recogmodelno]);
       i1:=1;t1:=1;t2:=1;
       pretest(recogmodelno,i1,t1,t2);
          if t1<>t2 then calculate_coord(recogmodelno,
                                         oritx,ority)
          else pretest(recogmodelno,i1+1,t1,t2);
       deletemodelno[recogmodelno]:= recogmodelno;
      end
   end;
end;

begin{main}
interest;
recognized;
end.
```

LIST OF REFERENCES

1. Lamdan, Y., Schwartz, J.T., and Wolfson, H.J., "Object Recognition by Affine Invariant Matching," *IEEE Conference on Computer Vision and Pattern Recognition*, June, 1988.

2. Lu, S.Y., and Fu, K.S., "Stochastic Error-Correction Syntax Analysis for Recognition of Noisy Patterns," *IEEE Trans. Comput.*, v. C-26, p. 296-301, December, 1977.

3. Fu, K.S., *On Syntactic Pattern Recognition and Stochastic Languages in Frontiers of Pattern Recognition*, pp. 56-78, S. Watanabe, editor, Academic Press, New York (1970).

4. Stockman, G.C. Recognition Parts and Their Orientation for Automatic Matching Machining, Handling and Inspection. Rep. NSF-SIBR-Phase I, NTIS Order PB 80-178817.

5. Wallace, T.P., Matchell, O.R., and Fukunaga, K., "Three Dimensional Shape Analysis Using Local Shape Descriptors," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. PAM-3, pp. 1004-1008, 1981.

6. Gonzalez, R.C, and Wintz, P., *Digital Image Processing*, p. 354, Addison-Wesley Publishing Company, 1977.

7. Ballard, D.H., "Generalizing the Hough Transform to Detect Arbitrary Shapes," in Pattern Recognition, v. 13, no. 2, pp. 111-122, 1981.

8. Moffitt, F.H., and Mikail, E., *Photogrammetry*, p. 593, Harper & Row Publishers, New York, 1980.

9. Lum, V.Y., Yuen, P.S.T., and Dodd, M., "Key-to-Address Transform Techniques: A Fundamental Performance Study on Large Existing Formatted Files," *Communications of the ACM*, April 1971, pp. 228-239.

10. *Subroutine Package for Image Data Enhancement and Recognition*, University of Osaka, Japan, 1983.

11. Pavlidis, T., *Algorithms for Graphics and Image Processing*, pp. 144-145, Computer Science Press, Inc., 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4.	Professor Chin-Hwa Lee, Code 62LE Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	7
5.	Professor Chang Yang, Code 62 YA Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	2
6.	Mr. Hsu, Tao-i Post Office Box 8243-19 TA KI, TAIWAN, R.O.C.	2

Thesis
H82975 Hsu
c.1 Affine invariant object
recognition by voting
match techniques.

Thesis
H82975 Hsu
c.1 Affine invariant object
recognition by voting
match techniques.

AMERICAN BINDERIES

Affine invariant object recognition by v
3 2768 000 81610 2
DUDLEY KNOX LIBRARY